About ObjectPAL

ObjectPAL is the object-based, event-driven, visual programming language for Corel Paradox. It is different from traditional procedural languages in many ways. You can use ObjectPAL to place objects (such as buttons and fields) in a form and to attach code modules (called methods) that execute whenever the object detects an event.

Object PAL, the programming language for Corel Paradox for DOS, is interpreted and its code is saved in script files. ObjectPAL is fully compiled, and its compiled code is stored in Windows DLLs along with its source code.

ObjectPAL has two components:

- The language itself (its object types, data types, methods, procedures, and constructs)
- The Integrated Development Environment (IDE), which includes

  The Editor

  The Debugger

  A mechanism for creating and playing ObjectPAL scripts

  The facilities for application delivery

There are six ObjectPAL language categories: data model objects, system data objects, data types, design objects, display managers, and events. Each category is in turn divided into several types.

In addition to these six categories, there are Basic Language Elements that are common to all methods and procedures.

The number of methods displayed in the Object Explorer for each type depends on which level of ObjectPAL you're working in: Beginner or Advanced. (All methods are *available* in both levels, but more complicated methods are only *displayed* in Advanced level.) To change the ObjectPAL level in the Developer Preferences dialog box, click Tools, Settings, Developer Preferences.

You can copy and paste the examples into your own code through the Clipboard.

- To copy the entire example to the Clipboard, right-click in the ObjectPAL Example window, and choose Copy.
- To copy a selected part of the example to the Clipboard, highlight the block of code you want in the Example window, right-click and choose Copy.
- Place the cursor in your code where you want to insert the example, then click Edit, Paste.

{button ,AL(`INTRO;',0,"Defaultoverview",)} Related Topics

About programming tasks

Here is a road map to the ObjectPAL language. Before you jump to any of these topics, make sure you read about Objects, Methods, and Events.

Messages and dialog boxes	Messages and built-in dialog boxes give you a way to interact with a user.
Handling keyboard events	You can trap for any keystroke in ObjectPAL, which means you can easily develop hotkeys for your application.
Working with menus	Using ObjectPAL, you can define menus and pop-up menus to display choices to users.
Working with lists	You can use list boxes and edit boxes to let a user choose from a group of items.
Multiform applications	To design applications that use more than one form, you'll need to know how to open a form and control it from another form. Forms can also be opened as dialog boxes.
Working with text files	You can use ObjectPAL to work with text files. Text files are called TextStreams in ObjectPAL.
Using DLLs	With the Uses clause, you can declare and subsequently use functions called from DLLs (dynamic link libraries).
Working with the file system	Using methods in the FileSystem type, you can access and get information about disk files, drives, and directories. ObjectPAL's fileBrowser procedure lets you display the Corel Paradox File Browser.

{button ,AL(`INTRO_PROGRAMMING;INTRO_INTRO;',0,"Defaultoverview",)} Related Topics

Messages and dialog boxes

Dialog boxes allow you to interact with Corel Paradox. System type procedures display dialog boxes (e.g., **dlgAdd** and **dlgCopy**) that begin with the prefix *dlg*. Procedures that display messaging dialog boxes, such as **msgInfo** and **msgQuestion**, begin with the prefix *msg*. The former do not return values; the latter return values such as Yes or No, in mixed upper and lowercase. You can also design your own dialog boxes.

A dialog box in Corel Paradox is really a form whose properties are set to make it act like a dialog box. To see the Window Style dialog box, create a new form and click Format, Window Style. Here you can add scroll bars and a standard menu, specify whether the dialog box is modal, and more. You can even open a normal form as a dialog box with the **openAsDialog** method. If you open a form from another form, you use the **wait** method to suspend execution in the first form until a **formReturn** method closes the second form.

Dialog boxes can be informational (e.g., displaying a simple message) or complex (e.g., prompting the user to enter search criteria for a query). There are many examples of how dialog boxes function in the *dlg* and *msg* topics. Also, the **message** procedure makes it easy to display up to six strings in the Status Bar.

{button ,AL(`OPAL_METH_SYDLGCREATE;OPAL_METH_SYDLGDELETE;OPAL_METH_SYDLGEMPTY;OPAL_METH_SYDLGEXPORT;OPAL_METH_SYDLGEXPORT;OPAL_METH_SYDLGIMPORTASCIIVAR;OPAL_METH_SYDLGIMPORTSPREADSH;OPAL_METH_SYDLGNETDRIVERS;OPAL_METH_SYDLGNETLOCKS;OPAL_METH_SYDLGNETREFRESH;OPAL_METH_SYDLGNETRETRY;OPAL_METH_SYDLGNETSETLOCKS;OPAL_METH_SYDLGNETSYSTEM;OPAL_METH_SYDLGNETUSERNAME;OPAL_METH_SYDLGNETWHO;OPAL_METH_SYDLGRENAME;OPAL_METH_SYDLGRESTRUCTURE;OPAL_METH_SYDLGSORT;OPAL_METH_SYDLGSUBTRACT;OPAL_METH_SYDLGTABLEINFO;OPAL_METH_SYMARI;OPAL_METH_SYMRCAOPAL_METH_SYMARI;OPAL_METH_SYMSTP;OPAL_METH_SYMYNC;INTRO_PROGRAMMING;',0,"Defaultoverview",)} **Related Topics**

Handling keyboard events

Keyboard events occur whenever you enter data at the keyboard or a keystroke autorepeats. Corel Paradox provides two built-in event methods for capturing KeyEvents: **keyChar** and **keyPhysical**.

Whenever you press a key, the event first goes to the form, which processes it if possible (e.g., when F1 is pressed) or dispatches it to the active object. The active object's built-in code then calls the **keyPhysical** method. If the keystroke represents an action to be performed, **keyPhysical** calls the **action** method. Otherwise, **keyPhysical** calls **keyChar**, which displays the character in a field on the active object. (In edit mode, **keyChar** first locks a field before it inserts a character.) If **keyChar** receives a SPACEBAR press when the active object is a button, it triggers the object's **pushButton** method.

You can use **keyPhysical** to create hot keys, which are keystroke equivalents of pressing buttons on a form.

{button ,AL(`OPAL_METH_SYSENDKEYS;OPAL_BMETH_LIST;OPAL_TYPE_KEYEVENT;INTRO_PROGRAMMING;',0,"Defaultoverview",)} Related Topics

Working with menus

Corel Paradox provides three types of menus.

Built-in menus and toolbars are complete and require no ObjectPAL coding. There may be times, however, when you want to disable menu options or add functionality to the default menus. Custom menus let you create menu designs for your applications.

If you decide to create a custom menu for your application, first decide where to put your ObjectPAL code. For example, if the menu will be available from the entire form, you might put the code in the form's **open** method. In a multi-page form, where you want different menus for different parts of the form, you might want to put your code in the page's **arrive** method. Next, decide where you want standard menus, which display across the top of the screen, and popup menus. You can then design the layout for your menus. The ObjectPAL Menu Type provides methods for creating and displaying menus, and for receiving user input.

{button ,AL(`OPAL_BMETH_MENUACTION;OPAL_TYPE_MENU;OPAL_TYPE_MENUEVENT;OPAL_TYPE_POPUPMENU;INTRO_PROGRAMMING;',0,"Defaultoverview",)} Related Topics

Working with lists

Lists and list boxes present a group of options or values from which the user selects. Lists are compound objects with two parts: the field object that you place on the form and the list object that contains the data shown in the list. You place a field object on the form and set its DisplayType property either to List or Drop-down Edit.

In a form, place a field object whose DisplayType property is set to Drop-down Edit or List. Click OK in the Define List dialog box. Choose Tools, Object Tree to view the relationship between the field object and the list object. You attach code, using the Object Explorer, to the field object that affects values displayed in the field and to the list object that affects values displayed in the list. Use the DataSource property to specify the source table and field whose data appears in the list.

You use a TCursor and its various methods to search the table field for a particular value, to add the value to the table if it does not exist, to edit the value, and more.

The following List properties that are particularly useful:

- list.count shows the number of items in a list. Setting this value to zero clears the list.
- list.selection sets the current index pointer in the list.
- list.value sets the value of the item currently pointed to by list.selection.

{button ,AL(`INTRO_PROGRAMMING;',0,"Defaultoverview",)} Related Topics

Multi-form applications

You may want to create applications that use more than one form or dialog box (which is a special kind of form). Use multiple forms sequentially when one task must be completed before another begins. Use multiple forms simultaneously to divide a complex application into functional modules.

Before you create a complex multi-form application, decide how you want its forms and dialog boxes to interact. Dialog boxes are either modal or non-modal. Typically, you use modal dialog boxes in sequential applications; that is, whenever user input is required before the application continues. Modality prevents the focus from changing to other windows, forms, system menus, and Windows applications. You use non-modal dialog boxes and standard forms in simultaneous applications; that is, whenever you want the user to have simultaneous access to other windows and resources.

How can you tell the difference between the behavior of modal and a non-modal dialog boxes? A modal dialog box cannot be resized. You must respond to the dialog box, either by typing in data, clicking a button, or closing the dialog box, before you can change focus to another object. A password dialog box is usually modal; you either enter a valid password or exit.

A non-modal dialog box can be resized, and it allows you to change focus to another window. A text search dialog box that lets you change focus to the underlying document is typical. Both modal and non-modal dialog boxes always rest on top of open forms, and both types of dialog boxes can be moved on top of the Toolbar.

ObjectPAL provides predefined dialog boxes; the System Type procedures that you use to display them begin with the prefixes *msg* or *dlg*. You can also design your own dialog boxes. You can even open a standard form as a dialog box with the openAsDialog method by specifying display attributes with WindowStyles constants.

The FormType methods, especially **wait**, **close**, and **formReturn** give you power and flexibility in handling the interaction of multiple forms and dialog boxes in applications.

{button ,AL(`OPAL_METH_SYDLGADD;OPAL_METH_SYDLGCOPY;OPAL_METH_SYDLGCREATE;OPAL_METH_SYDLGDELETE;OPAL_METH_SYDLGEMPTY;OPAL_METH_SYDLGEXPORT;OPAL_METH_SYDLGEXPORT;OPAL_METH_SYDLGIMPORTASCIIVAR;OPAL_METH_SYDLGIMPORTSPREADSH;OPAL_METH_SYDLGNETDRIVERS;OPAL_METH_SYDLGNETLOCKS;OPAL_METH_SYDLGNETREFRESH;OPAL_METH_SYDLGNETRETRY;OPAL_METH_SYDLGNETSETLOCKS;OPAL_METH_SYDLGNETSYSTEM;OPAL_METH_SYDLGNETUSERNAME;OPAL_METH_SYDLGNETWHO;OPAL_METH_SYDLGRENAME;OPAL_METH_SYDLGRESTRUCTURE;OPAL_METH_SYDLGSORT;OPAL_METH_SYDLGSUBTRACT;OPAL_METH_SYDLGTABLEINFO;OPAL_METH_SYMESSAGE;OPAL_METH_SYMARI;OPAL_METH_SYMRCAOPAL_METH_SYMARI;OPAL_METH_SYMSTP;OPAL_METH_SYMYNC;INTRO_PROGRAMMING;',0,"Defaultoverview",)} **Related Topics**

Working with text files

To work with ANSI text files, you use the TextStream data type. The TextStream type includes all ANSI characters, including such non-printing characters as the carriage return and line feed. To work with formatted text files, which include such attributes as font, alignment, and margins, use the Memo type.

There are TextStream methods for such tasks as opening and closing files, reading one line or one character of text at a time, reading from and writing to disk, and setting the position of the pointer within a file. A TextStream pointer shows the current position in the file counted from the beginning: 1 is the first character, 2 the second, and so on. The TextStream **advMatch** method searches a file for a pattern, similar to the String **advMatch** method.

TextStreams and Strings are related objects that both contain text; however, only TextStreams, can read from and write to files on disk.

{button ,AL(`INTRO_PROGRAMMING;',0,"Defaultoverview",)} Related Topics

Using DLLs

Dynamic Link Libraries (DLLs), store functions in a library that is external to Corel Paradox and ObjectPAL. Before you call a custom, external routine from ObjectPAL, you first declare it in a **uses** clause at the beginning of your method or procedure, or in the Uses window available from the Corel Paradox Object Explorer.

It is best to use Corel Paradox and ObjectPAL methods and procedures when you build applications because they are efficient and powerful. You create a DLL only when you want to add to or enhance Corel Paradox capabilities. For example, you may want to call routines in COMMDLG32.DLL to use a Windows common dialog box, or in USER32.EXE to use other Windows functions. If you have an existing application that uses 16-bit DLLs or 16-bit Windows calls, you need to replace the DLLs and Win Application Programming Interface (API) calls with 32-bit versions.

Example

{button ,AL(`INTRO_PROGRAMMING;',0,"Defaultoverview",)} Related Topics

Using DLLs example

The following example calls the 32-bit Windows DLL, WINMM.DLL, and its function, PlaySound, to play the Microsoft Sound Wav file when the button is pressed.

The first set of code goes into the button's Uses method. The Function Names in the Uses block are case-sensitive.

```
Uses WINMM

PlaySound(WavFileName CPTR,How CWORD, Flag CWORD)

endUses
```

The following code is inserted into the button's **pushButton** event, and invokes the PlaySound function with the necessary parameters:

```
Method pushButton(var eventInfo Event)

playsound("C:\\Windows\\media\\The Microsoft Sound.wav",0,0)

endMethod
```

Working with the file system

You use the FileSystem data type to work with disk files, drives, and directories. FileSystem variables provide a handle for access to files. The first step is often to call the **findFirst** method for a FileSystem variable, which checks whether a file or directory exists; if so, **findFirst** initializes the variable with a handle to the file or directory. You then use the FileSystem methods to work with drives, directories, and files.

Corel Paradox provides a File Browser for performing interactive file system tasks. The ObjectPAL System Type provides the fileBrowser procedure, which displays the Corel Paradox FileBrowser.

{button ,AL(`INTRO_PROGRAMMING;',0,"Defaultoverview",)} Related Topics

ObjectPAL language categories

Corel Paradox and ObjectPAL let you create compiled applications from these major components:

Category	Description	Object types
Data model	Let you work with data in tables	Database, Query, Table, TCursor, SQL
System data	Let you store data, but not in tables	DataTransfer, DDE, FileSystem, Library, Mail, Session, System, TextStream
Data types	The basic ObjectPAL data types	AnyType, Array, Binary, Currency, Date, DateTime, DynArray, Graphic, Logical, LongInt, Memo, Number, OLE, Point, Record, SmallInt, String, Time
Design objects	Let you create the user interface to your application	Menu, Native Windows Controls, PopUpMenu, UIObject, Toolbar
Display managers	Let you control how data is presented to the user	Application, Form, Report, Script, TableView
Event Types	Contain information about actions in Corel Paradox	ActionEvent, ErrorEvent, Event, KeyEvent, MenuEvent, MouseEvent, MoveEvent, StatusEvent, TimerEvent, ValueEvent

Corel Paradox and ObjectPAL build applications incrementally. First, you build tables, forms, and objects in Corel Paradox, and add custom ObjectPAL methods to alter their default methods. Once the objects have been tested and debugged, you can refine the application by adding more objects.

Recent versions of Corel Paradox allow you to control objects that are native to the Windows operating system. Native Windows controls (NWCs) allow Corel Paradox forms to deliver information to the user in the same format as Windows.

{button ,AL(`INTRO_CONCEPTS;INTRO_INTRO;',0,"Defaultoverview",)} Related Topics

Objects

In Corel Paradox, most of the things you work with are objects—the buttons and fields you create using the Toolbar, the tables and text files stored on disk, and the menus created in code, to name a few. Corel Paradox recognizes two kinds of objects: design objects and data objects. You place design objects, such as buttons, list boxes, and other UIObjects, in forms. Data objects are files, data types, and programming structures.
All Corel Paradox objects have properties (attributes such as color, font, and line width) and methods (code that defines how they respond to an event),

Most objects (except for ActiveX controls) that you can create or modify interactively using Corel Paradox, can also be creates or modified using ObjectPAL.

Objects and methods

Custom Corel Paradox applications are created by placing objects in forms and then writing ObjectPAL methodsto define how those objects respond to events. Such applications are sometimes called "Hey you, do this" applications. The "Hey you" part (called an event) happens when the user does something to an object (e.g., clicks a button on a form). The "do this" part is defined by methods or code that executes when an event occurs. Some objects can display other objects (such as a lookup list) or chain to another stage in the application (e.g., another form, query, or report).

Object types

ObjectPAL objects are grouped by type. ObjectPAL Language Categories contain groups of types and give you access to help on the methods in each type.

Syntax, description, and sample code is provided for each method. You can copy and paste sample code into your own code through the Clipboard.

Properties

Each type of object has properties or attributes such as focus, value, name and visible. For example, buttons have the ButtonType, CenterLabel, and Name properties; box objects have the Color, Frame.Color, and Size properties; and forms have DialogForm, HorizontalScrollBar, and SnapToGrid properties.

Events

Corel Paradox recognizes certain actions or conditions within forms as events. When Corel Paradox detects that an event has occurred, it triggers execution of the method associated with that event. There are different types of events that are appropriate for different types of objects. For instance, the pushButton event is recognized by a pushButton object, but not by a graphic object. There are, however, events that are recognized by most or all objects, such as the timer event, events related to focus, and opening and closing.

To understand how Corel Paradox processes events, you must understand bubbling, the process by which events pass from the target object up through the containership hierarchy. When an event occurs, the target object does not immediately process the event. Instead, the event passes up to the form level, where the form determines whether to process the event, to send the event back to the target object, or to another object for processing. The object that finally processes the event triggers the method (code) associated with the event.

{button ,AL(`INTRO_CONCEPTS;INTRO_INTRO;',0,"Defaultoverview",)} **Related Topics**

Events

Examples of events are:

- pressing the mouse button
- releasing the mouse button
- moving the pointer over an object
- pressing a key
- moving the cursor into a field
- moving the cursor out of a field
- selecting an item from a menu

Events can also happen for other reasons; for example, the timer event happens after a certain amount of time passes. You can also generate events from within your own methods.

Using ObjectPAL, you can create methods that define how objects respond to events. All objects have default methods for ObjectPAL events. You don't have to write methods for all the events an object can handle, and an event never goes unrecognized.

{button ,AL(`INTRO_CONCEPTS;INTRO_INTRO;',0,"Defaultoverview",)} Related Topics

Properties

Objects have properties such as color, pattern, font, and line width.

You can use Corel Paradox to set and change these properties or you can use ObjectPAL. Almost everything you can do in Corel Paradox, you can do in ObjectPAL.

For example, the following statements set the color of rectangle *box1* to red, set the font of field *field1* to Times, and make *myCircle* invisible.

```
box1.color = "Red"    ; sets color of box1 to red
field1.font = "Times" ; sets font of field1 to times
myCircle.visible = No
```

{button ,AL(`INTRO_CONCEPTS;INTRO_INTRO;',0,"Defaultoverview",)} **Related Topics**

List of data types

The ObjectPAL language contains the following data types:

AnyType	LongInt
Array	Memo
Binary	Number
Currency	OLE
Date	Point
DateTime	Record
DynArray	SmallInt
Graphic	String
Logical	Time

{button ,AL(`INTRO_CONCEPTS;INTRO_INTRO;',0,"Defaultoverview",)} Related Topics

Containership

Corel Paradox objects coexist in a hierarchy of containership; for example, when you place a table object on a page of a form, that page **contains** the table. Forms contain tables, tables contain records, records contain fields, fields can contain buttons, and so on.

An object is contained only if it is completely within the boundaries of the container.

Position in this hierarchy is important because it defines whether an object can see another object's properties and variables.

An object cannot see variables in the objects it contains, but it can see its own variables, and the variables in the objects that contain it. To put it another way, if you think of objects as boxes that contain smaller boxes, the smallest box has the best view.

{button ,AL(`INTRO_CONCEPTS;INTRO_INTRO;',0,"Defaultoverview",)} **Related Topics**

About procedures

There are two kinds of procedures in ObjectPAL: procedures in the ObjectPAL run-time library (RTL) and custom procedures you create. Procedures in the RTL are just like methods except they never explicitly specify an object. A custom procedure resembles procedures in many other programming languages; it is a routine you write yourself and use like a subroutine.

{button ,AL(`INTRO_COMPONENTS;INTRO_COMPONENTS_PROC;',0,"Defaultoverview",)} Related Topics

RTL procedures

The procedures in the ObjectPAL run-time library are just like ObjectPAL methods, with one exception. Procedures never specify an object. Any method in any object can call any ObjectPAL procedure, and the procedure will know what to do. For example, the statement

```
close()
```

calls the Form type procedure **quit**, which closes the current form. The System type includes a number of procedures to interact with users (e.g., **message**, **msgInfo**, and **msgStop**).

```
msgStop("Alert!", "This file already exists.")
```

The System type also includes the procedures **beep** and **sleep** and several enumeration procedures for getting and setting the mouse position and shape:

```
method pushButton(var eventInfo Event)
beep()              ; plays the system beep sound
sleep(2000)         ; waits for 2 seconds
beep()
message("Did you hear two beeps?")
                    ; displays a message in the status line
sleep(2000)
enumAllObjectSource("mySource.db")
                    ; creates a table of all methods in this form
endMethod
```

Like ObjectPAL methods, ObjectPAL procedures are associated with object types and execute in response to events. It may be helpful to think of ObjectPAL procedures as methods in which the object is implied.

{button ,AL(`INTRO_COMPONENTS_PROC;',0,"Defaultoverview",)} Related Topics

Custom procedures

Custom procedures in ObjectPAL resemble procedures in many other programming languages. A custom procedure is a routine you write and use like a subroutine.

Custom procedures can be attached to the object itself, to any object in the containership hierarchy, or to the form itself.

Custom procedures can be included in libraries, but they can only be invoked from within the library.

ObjectPAL can call a custom procedure faster than it can call a custom method. The code executes at the same speed, but ObjectPAL can find a procedure faster than it can find a method.

Use a **proc** block to declare a custom procedure. The structure is

```
PROC name (parameterDescription) [return type]
   [CONST section]
   [TYPE section]
   [VAR section]
   [ObjectPAL statements]
ENDPROC
```

You can declare procedures in two places:

- within a method
- in an object's Proc window

{button ,AL(`INTRO_COMPONENTS_PROC;',0,"Defaultoverview",)} Related Topics

Procedures declared in methods

A procedure declared in a method is private. Its scope is limited to the method in which it is defined.

Here's an example of a custom procedure:

```
proc inc(x SmallInt) SmallInt
   return x+1 ; increments a number
endProc
```

The following example shows how to call that procedure (and another one) from within a method. In this example, it's the **pushButton** method, but it could be any method:

```
proc inc(x SmallInt) SmallInt
   return x+1
endProc


proc showMe(x SmallInt)
   msgInfo("myNum = ", x)
endProc


method pushButton(var eventInfo Event)
var
   myNum SmallInt
endVar
   myNum = 3
   showMe(myNum)
   myNum = inc(myNum)
   showMe(myNum)
endMethod
```

{button ,AL(`INTRO_COMPONENTS_PROC;',0,"Defaultoverview",)} Related Topics

Procedures declared in an object's Proc window

A procedure declared in an object's Proc window has the same syntax as a procedure declared in a method, but it has a different scope.

A procedure declared in an object's Proc window is visible both to all methods attached to that object and to all methods in objects contained by that object. Therefore, to make a procedure available to every object in a form, declare the procedure in the form's Proc window.

{button ,AL(`INTRO_COMPONENTS_PROC;',0,"Defaultoverview",)} Related Topics

About methods

A method is code that defines the behavior of an object in response to events. ObjectPAL methods fall into one of three categories:

- built-in event methods that are included with every Corel Paradox object
- methods in the ObjectPAL run-time library
- custom methods that you create

{button ,AL(`INTRO_COMPONENTS;INTRO_COMPONENTS_METH;INTRO_COMPONENTS_METH_EV;',0,"Defaultoverview",)} Related Topics

Editing a method

You can edit a method for an object in a form, or for the form itself.

To edit a method for an object

1. Open the Object Explorer.
2. Click the Methods Tab.
3. Double-click the method you want to edit.

An Editor window opens.

To edit the method for a form

1. Open the Object Explorer.
2. Select the Form icon at the top of the object tree
3. Click the Methods Tab.
4. Double-click the method you want to edit.

An Editor window opens.

You can type the text for a method directly in the Editor, or use the Clipboard to copy, cut, and paste methods and parts of methods from other objects. However, there is no linkage or relationship between the original method and the copied method. Changes made to one method are not reflected in the other.

You can also copy a method by copying an object from a design document. When you copy an object, all methods attached to the object are copied as well. However, there is still no linkage between the methods after the copy.

{button ,AL(`INTRO_COMPONENTS_METH;INTRO_COMPONENTS_METH_EV;',0,"Defaultoverview",)}
Related Topics

Methods in the run-time library

The ObjectPAL run-time library (RTL) is a collection of predefined routines. The library includes methods you can use to perform a wide range of tasks, from reading and editing data in tables to creating and displaying menus. Each of these methods is associated with an object type; all the methods for working on forms are in the Form type, all the methods for working with text files are in the TextStream type, and so on.

ObjectPAL methods are symmetrical and consistent. Within a type, methods often come in pairs. For example, if a type has an **open** method, you can expect it to have a **close** method. If you can read information from an object, you can write to it; if you can get a value, you can set a value.

ObjectPAL methods are consistent across types because methods with similar names do similar things. For example, **open** makes an object available for manipulation, whether the object is a table or a text file, and **close** puts it away. The underlying code may differ, but conceptually, the results are the same.

Methods in the run-time library require you to use dot notation to specify an object on which to operate.

{button ,AL(`INTRO_COMPONENTS_METH;INTRO_COMPONENTS_METH_EV;',0,"Defaultoverview",)}
Related Topics

Custom methods

Custom methods are auxiliary methods you create. They are convenient for making frequently used routines available to several objects.

Custom methods attached to a form are available to all objects in the form. That way, you only have to maintain the code in one place.

After you save a custom method, its name is listed in the Object Explorer. To make changes, choose the name and open an ObjectPAL Editor window, just as you would to edit a built-in event method.

{button ,AL(`INTRO_COMPONENTS_METH;INTRO_COMPONENTS_METH_EV;',0,"Defaultoverview",)}
Related Topics

Creating a custom method

To create a custom method

1. Open the Object Explorer.
2. Click the Methods tab.
3. Double-click New Method.
4. Type the name of the new custom method.
5. Click OK to open an Editor window for the new method.

You can type or paste text into custom methods just as you can for built-in event methods.

After you save a custom method, its name is listed in the Object Explorer. To make changes, choose the name and open an ObjectPAL Editor window, just as you would to edit a built-in event method.

You can copy, cut, and paste an entire object. When you do, all methods attached to the object are copied as well. However, there is no link or relationship between the original method and the copied method. Changes made to one are not reflected in the other.

Methods in other objects

Methods are public. Methods attached to an object can be called by other objects. For example, suppose a form contains two boxes: *box1* and *box2*. If *box1* has a method **fred**, *box2* could use dot notation to call it:

```
box1.fred()
```

If you attach a custom method to a form—the top level of the containership hierarchy—all objects contained in the form have direct access to that method. For example, if you attach the custom **goNextPage** method to a form, a button on that form could call **goNextPage** like this:

```
method pushButton (var eventInfo Event)
goNextPage() ; this is a custom method attached to the form
endMethod
```

In this example, we didn't have to use dot notation because the **pushButton** method is attached to the button and the button is contained by the form; therefore, the button has direct access to the form's methods.

When you compile this method, ObjectPAL searches other objects for **goNextPage**, so it executes without delay at run time.

{button ,AL(`INTRO_COMPONENTS_METH;INTRO_COMPONENTS_METH_EV;',0,"Defaultoverview",)}
Related Topics

Method language structure and syntax

In terms of structure and syntax, ObjectPAL methods resemble traditional programs. Some aspects of this structure are:

- Methods can have parameters (also called arguments).
- Methods are delineated by the **method...endMethod** keywords. You can define an ordered structure of execution because ObjectPAL supports control structures and loops like **while...endWhile**, **if...then...else...endIf**, and **switch...case...endSwitch**.
- As in Pascal and C, you can define procedures to perform one or more tasks. Procedures can receive arguments from, and return results to, the method that calls them.
- Also as in C, you can freely use white space (tabs, spaces, and blank lines). You can choose to indent subordinate method lines, put one or more statements on a line, and append a comment to any method line—white space has no effect on how statements are executed.

{button ,AL(`INTRO_COMPONENTS_METH;INTRO_COMPONENTS_METH_EV;',0,"Defaultoverview",)}
Related Topics

Variables

A variable is like a slot where you can temporarily store one item of information.

The value of a variable can be any ObjectPAL type (also called a data type). It is not necessary to explicitly indicate a data type for variables.

When you specify a variable's data type before you use it is called declaring a variable.

The simplest way to give a variable a value is to use the assignment operator (=).

{button ,AL(`INTRO_VARIABLES;INTRO_INTRO;',0,"Defaultoverview",)} **Related Topics**

The scope of a variable

The term scope means accessibility. The scope of a variable, that is, the range of objects that have access to it, is defined by the objects in which it is declared and by the containership hierarchy. Objects can only access their own variables and variables defined in the objects that contain them. The scope of a variable also depends on where it is declared.

Within a method

Variables declared within a method are visible only to that method, and they are accessible only while that method executes. They are initialized (reset) each time the method executes.

Outside a method

Variables declared in a Method window before the keyword **method** are visible only to that method, but they are not initialized each time the method executes.

In the Var window

Variables declared in an object's Var window are visible to all methods attached to that object and to any objects that the object contains. A variable declared in an object's Var window is attached to the object, and the variable is accessible as long as the object exists in the form and the form is open.

Within the containership hierarchy (compile-time binding)

In programming terms, binding a variable is connecting a variable to a data type. The ObjectPAL compiler binds variables when it compiles the source code; there is no run-time binding in ObjectPAL. When the compiler encounters a variable in a statement, it searches the rest of the source code to find out where the variable is declared so it can bind the variable to the declared data type.

{button ,AL(`INTRO_VARIABLES;',0,"Defaultoverview",)} Related Topics

Constants

The ObjectPAL language includes many predefined constants. Constants are like variables except they are protected from change when the program runs. This enables the compiler to generate more efficient code.

You can define constants for a single method, or open a Const window to define constants for all the object's methods.

Constants are automatically put into resources, where they can be modified without affecting the source code.

{button ,AL(`INTRO_VARIABLES;INTRO_INTRO;',0,"Defaultoverview",)} **Related Topics**

Introduction to scripts

A script consists of code in its own file. It is not attached to a form. A script is an object that displays on the desktop as an icon. Use a script when you want to execute code without opening and displaying a form window. You can

- attach code to one or more built-in event methods
- declare variables, constants, data types, custom methods, and procedures
- call custom DLLs

A script does not display in a window and does not contain any design objects. A script has the built-in event methods **run**, **error**, and **status**. (You must set your ObjectPAL Level to Advanced in the General page of the Developer Preferences dialog box to display the **status** method in the list of built-in event methods.) You can execute these methods interactively using Corel Paradox, or call them from within an ObjectPAL method or procedure. Like any other object, a script also has windows for declaring variables, constants, procedures, data types, and external routines. You can also declare custom methods.

From a script, you have complete access to the ObjectPAL run-time library; therefore, you can control other objects. For example, you can call other scripts, open and work with tables, forms, and reports, and run queries. You can call methods attached to other objects and get and set their properties.

A Script type includes methods for creating and manipulating scripts—and the code they contain—from within an ObjectPAL method or procedure.

{button ,AL(`INTRO_SCRIPTS;INTRO_INTRO;',0,"Defaultoverview",)} Related Topics

Creating a script

To create a script

1. Click File, New to open the New dialog Box.
2. Double-click the New Script icon on the Create New page. An ObjectPAL Editor window opens for the Script's built-in **run** method.
3. Type your code in the window. This is a standard ObjectPAL Editor window. You can edit, check syntax, and debug the **run** method as you would any other object. Keep in mind that whatever you declare is visible only to the script's **run** method.
4. Close the window and type a name for the script.

Like a form, a script can be saved or delivered. Saved scripts can be changed; delivered scripts cannot.

You can also create a script using ObjectPAL. For more information, see Script Type.

{button ,AL(`INTRO_SCRIPTS;',0,"Defaultoverview",)} Related Topics

Adding new code to a script

To add code to a script

1. Open the script in an Editor window.
2. Type your code in the window.
3. To save the script (both source code and executable code) to disk, click File, Save. To save only the executable code, click Program, Deliver.

Using the Object Explorer and ObjectPAL Editor windows, you can add code to a script by

- attaching code to the built-in event methods
- adding custom methods
- adding custom procedures
- declaring variables, constants, data types, and external routines

You can also add code to a script using ObjectPAL. For more information, see Script Type.

{button ,AL(`INTRO_SCRIPTS;',0,"Defaultoverview",)} Related Topics

About attaching code to built-in event methods (in scripts)

Every script has the following built-in event methods: **run**, **error**, and **status**. (You must set your ObjectPAL Level to Advanced to display **status** in the list of built-in event methods.) You can attach code to these built-in event methods as you would with any other object. You do this in the Editor window.

You can also attach code to a script's built-in event methods using ObjectPAL. For more information, see Script Type.

{button ,AL(`INTRO_SCRIPTS;',0,"Defaultoverview",)} Related Topics

Adding custom methods to scripts

You can add custom methods to a script using the Object Explorer.

To add a custom method

1. Open the script in an Editor window.
2. Click Tools, Object Explorer.
3. Click the Methods tab.
4. Double-click New Method.
5. Type the name of the new custom method.
6. Click OK to open an Editor window for the new method.

You can also add custom methods to a script by using ObjectPAL. For more information, see Script Type.

{button ,AL(`INTRO_SCRIPTS;',0,"Defaultoverview",)} Related Topics

Adding custom procedures

From a script's Object Explorer, you can choose Proc to open an Editor window in which you can declare custom procedures for the script.

You can also add custom procedure to a script using ObjectPAL. For more information, see Script Type.

{button ,AL(`INTRO_SCRIPTS;',0,"Defaultoverview",)} Related Topics

Declaring variables, constants, data types, and external routines

From a script's Object Explorer, you can declare variables, constants, data types, and external routines by choosing Var, Const, Type, or Uses, respectively, to open the appropriate Editor window. Items declared in these windows are global to the script but cannot be accessed by other forms or objects.

{button ,AL(`INTRO_SCRIPTS;',0,"Defaultoverview",)} **Related Topics**

Editing a script

You can edit a script in an ObjectPAL Editor window.

To edit a script

1. Click File, Open, Script.
2. Enable the Edit The Script button at the bottom of the Open Script dialog box
3. Double-click the script you want to open.

The script opens in an Editor window with the built-in event method **run** displayed.

4. Use the ObjectPAL Editor to edit your script as you would a method.

{button ,AL(`INTRO_SCRIPTS;',0,"Defaultoverview",)} Related Topics

Debugging a script

You can debug a script by using the ObjectPAL Debugger.

To debug a script

1. Open the script in an Editor window.
2. Set breakpoints and watch points as desired.
3. Run the script.

When ObjectPAL encounters the breakpoint, execution stops and the script opens in a Debugger window. Use the ObjectPAL debugger to debug the script as you would a method.

{button ,AL(`INTRO_SCRIPTS;',0,"Defaultoverview",)} Related Topics

About playing a script

You can play a script interactively using Corel Paradox or from within a method. In either case, the result is that you execute the script's built-in **run** method.

{button ,AL(`INTRO_SCRIPTS;',0,"Defaultoverview",)} **Related Topics**

Playing a script interactively

To play a script interactively

1. Click File, Open, Script.
2. Enable the Run The Script button at the bottom of the Open Script dialog box.
3. Double-click the name of the script you want to open.

The script's built-in **run** method executes.

{button ,AL(`INTRO_SCRIPTS;',0,"Defaultoverview",)} **Related Topics**

Playing a script programmatically

Use the System type method **play** to play a script from within a method or procedure. For example:

```
switch
   case theValue = "this" : play("doThis")   ; play script "doThis"
   case theValue = "that" : play("doThat")   ; play script "doThat"
   otherwise              : play("theOther") ; play script "theOther"
endSwitch
```

{button ,AL(`INTRO_SCRIPTS;',0,"Defaultoverview",)} **Related Topics**

Delivering a script

When you deliver a script, Corel Paradox removes all the source code. Your code is not lost; it is protected.

If you save the script using File, Save, anyone who uses the script can modify the ObjectPAL code, and change your application. Delivery gives you a way to let others use your code, but not change it.

To deliver a script you have created

- With the script open in an Editor window, click Program, Deliver.

When you click Program, Deliver, Corel Paradox saves a copy of the script with an .SDL extension. You can still change your ObjectPAL code by using the saved script (with the .SSL extension), but if you want others to use it safely, give them the delivered script.

{button ,AL(`iide_about_delivering_application;INTRO_SCRIPTS;',0,"Defaultoverview",)} Related Topics

Introduction to libraries

A library is a collection of custom methods and procedures. Libraries are used to store and maintain frequently used routines and to share custom methods and variables among several forms.

In many ways, working with a library is like working with a form. For example, a library has built-in event methods. You add code to a library just as you do to a form, by using the Object Explorer and the ObjectPAL Editor. As with a form, you can open Editor windows to declare custom methods, procedures, variables, constants, data types, and external routines.

However, there are some important differences:

- At run time, a library does not display in a window.
- A library cannot contain design objects; it can contain only code.
- In a Library, statements that use Self do not refer to the Library—instead, they refer to the object that called the method.
- The scoping rules are different for libraries.

{button ,AL(`INTRO_LIBRARIES;INTRO_INTRO;',0,"Defaultoverview",)} Related Topics

Library methods

You can use the Library methods in your own code—attached to any object, or even another library—to manipulate a library. See Library Type for a list of the run-time methods.

{button ,AL(`INTRO_LIBRARIES;',0,"Defaultoverview",)} **Related Topics**

Calling library methods

To call a method in a library, you must first declare the library in the Uses window of the object that is doing the calling. For example, suppose you want a button's **pushButton** method to call a custom method from a library. Declare the library in the button's Uses window (or in the Uses window of an object that contains the button). This tells Corel Paradox where to look for the method and what arguments it will take.

{button ,AL(`INTRO_LIBRARIES;',0,"Defaultoverview",)} Related Topics

Creating a library

To create a new library

1. Click File, New to open the New dialog Box.
2. Double-click the New Library icon on the Create New page.

A Library window opens.

3. Click Tools, Object Explorer.
4. Use the Object Explorer and ObjectPAL Editor windows to add code to a library by

- attaching code to the built-in event methods
- adding custom methods
- adding custom procedures
- declaring variables, constants, data types, and external routines

When you are finished adding code, you can save both the source code and the executable code (to an .LSL file) by clicking File, Save, or you can save only the executable code (to an .LDL file) by clicking Program, Deliver.

{button ,AL(`INTRO_LIBRARIES;',0,"Defaultoverview",)} Related Topics

Adding code to a library

You can add code to an existing library.

To add code to a library

1. Open the Object Explorer.
2. Use the Object Explorer and its ObjectPAL Editor windows to add code to a library by

- attaching code to the built-in event methods
- adding custom methods
- adding custom procedures
- declaring variables, constants, data types, and external routines

When you are finished adding code, you can save both the source code and the executable code (to an .LSL file) by clicking File, Save, or you can save only the executable code (to an .LDL file) by clicking Program, Deliver.

{button ,AL(`INTRO_LIBRARIES;',0,"Defaultoverview",)} Related Topics

Attaching code to built-in event methods (in libraries)

Every library has the following built-in event methods: **open**, **close**, and **error.** You can attach code to these built-in event methods, as you would with any other object, in the Editor window.

A library's built-in **open** method is called when the library is first opened; **close** is called when the library is being closed; **error** is called when code in the library generates an error. Typically, you will use **open** to initialize global library variables, and use **close** to tidy up after using the library. By default, a library's **error** method calls the **error** method of the form that called the library routine.

{button ,AL(`INTRO_LIBRARIES;',0,"Defaultoverview",)} Related Topics

Adding custom methods

The custom methods in a library can be called by other methods in the same library, by methods in other forms, and by methods in objects in other forms. This accessibility makes libraries very useful.

You can add custom methods from the Object Explorer window.

To add a custom method

1. Open the Object Explorer.
2. Click the Methods tab.
3. Double-click New Method.
4. Type in a name for the new custom method, and click OK.

An Editor window opens.

{button ,AL(`INTRO_LIBRARIES;',0,"Defaultoverview",)} Related Topics

Adding custom procedures

You can add custom procedures from the Object Explorer window.

To add a custom procedure

1. Open the Object Explorer.
2. Click the Methods Tab.
3. Double-click Proc.

An Editor window opens in which you can declare custom procedures for the library.

Note

- Unlike custom methods, which can be called from other forms and other objects, custom procedures can only be called from within the library in which they are declared.

{button ,AL(`INTRO_LIBRARIES;',0,"Defaultoverview",)} **Related Topics**

Declaring variables, constants, data types, and external routines

From a library's Object Explorer, you can declare variables, constants, data types, and external routines by choosing Var, Const, Type, or Uses, respectively, to open the appropriate Editor window. Items declared in these windows are global to the library, but are not available to other forms or objects. However, other forms and objects can call library routines that access these variables.

{button ,AL(`INTRO_LIBRARIES;',0,"Defaultoverview",)} Related Topics

Debugging called libraries

To debug ObjectPAL code on a form or library that is called from another form or library, you must set a breakpoint both in the ObjectPAL code that calls the second form or library, and in the code on the second form or library. While single-stepping through the code that calls the second form or library, step into the method or procedure on the second form or library.

{button ,AL(`INTRO_LIBRARIES;',0,"Defaultoverview",)} **Related Topics**

Editing a library

You can edit a library in an ObjectPAL Editor window.

To edit a library

1. Open the Object Explorer.
2. Click the Methods Tab.
3. Double-click the method you want to edit.

An Editor window opens for the method you selected. Use the ObjectPAL Editor to edit this method as you would any method.

{button ,AL(`INTRO_LIBRARIES;',0,"Defaultoverview",)} Related Topics

Delivering a library

When you deliver a library along with an application, Corel Paradox removes all the source code. Your code is not lost; it is protected.

If you save the library using File, Save, anyone who uses the library can modify the ObjectPAL code and change your application. Delivery gives you a way to let others use your code, but not change it.

To deliver a library

- With the Library window open, click Program, Deliver.

Corel Paradox saves a copy of the library with an .LDL extension. You can still change your ObjectPAL code by using the saved library (with the .LSL extension), but if you want others to use it safely, give them the delivered library.

{button ,AL(`INTRO_LIBRARIES;',0,"Defaultoverview",)} **Related Topics**

Controlling the scope of a library

The scope of a library refers to its accessibility (i.e., which objects have access to the library's code). A Library variable follows the same scoping rules as other ObjectPAL variables. Two things determine a library's scope: where the Library variable is declared, and how the library is opened.

{button ,AL(`INTRO_LIBRARIES;',0,"Defaultoverview",)} Related Topics

Declaring a Library variable

A Library variable follows the same scoping rules as other ObjectPAL variables. A variable declared in a method is available as long as that method is executing. A variable declared in an object's Var window is available both to the methods attached to that object, and to all objects that object contains.

To make a library available to all objects in a form for as long as that form is running, declare the Library variable in the form's Var window and declare the library routines in the form's Uses window.

{button ,AL(`INTRO_LIBRARIES;',0,"Defaultoverview",)} Related Topics

Using library variables as arguments

You can use a Library variable as an argument in a custom method or custom procedure.

By passing a library as an argument, you can change the behavior of a routine (method or procedure) and still maintain the routine's independence. A routine may use a library and routines from the library, but the caller can determine the function of the routines by just changing the library.

{button ,AL(`INTRO_LIBRARIES;',0,"Defaultoverview",)} Related Topics

Opening a library

The Library method **open** takes arguments that specify the scope. A library can be opened in either of the following ways:

- private to the form, so that only the form that opened the library has access to its code
- global to the desktop, so that every form on the desktop can access the library. This lets several forms access the same custom methods and share the same global variables. By default, a library opens global to the desktop.

For two or more forms to share the same library, each form must open the library global to the desktop, and each form must have a Uses window that declares which library routines to use.

{button ,AL(`INTRO_LIBRARIES;',0,"Defaultoverview",)} Related Topics

About the Object Explorer

The Object Explorer is your entryway to the ObjectPAL Editor. It lets you view an object tree for the current form and gives you a developer's interface to properties, which you can change in the Explorer.

The Object Explorer has two panes: an object tree pane and a tabbed pane that shows the object's appearance, methods, properties, and events. You can display the panes side-by-side or you can display each pane individually. These choices are available on the Object Explorer View menu. You can also adjust the size of the panes by dragging the border between the two panes.

The object tree displays information for the current document; the tabbed pane displays information for the selected object. If you select another object or document, the contents of the Object Explorer change to reflect the new object or document.

The two panes share four menus:

- the File menu, which is used to close the Object Explorer
- the Edit menu, which contains editing commands
- the View menu, which is used to specify what part of the Object Explorer to view, whether to hide the main menu, and whether to temporarily pin the Object Explorer to the desktop
- the Help menu, which is used to get help on the Object Explorer

The object tree

The object tree shows the hierarchical relationships among objects in the current form. It works like the Windows Explorer: click a plus (+) icon to expand that node; click a minus (-) icon to collapse it. When fully expanded, the object tree shows all objects on the current form. You can move and copy objects in the object tree using the right-click menu. You can also right-click the objects in the object tree to change their properties.

For information on using the object tree, see About the object tree.

The tabbed pane

The tabbed pane contains separate pages that show what appearance traits, properties, methods, and events are attached to an object. It lets you change the properties and appearance of an object and open individual Editor windows to edit methods and events.

For information on using the tabbed pane, see About the tabbed pane.

{button ,AL(`INTRO_EXPLORER;INTRO_INTRO;',0,"Defaultoverview",)} Related Topics

Opening the Object Explorer

You can open the Object Explorer from a Design, Script, or Library window. From the Object Explorer, you can use the object tree or the tabbed panes to find and edit the different elements of your document.

To open the Object Explorer from a form

1. With the form open, click View, Design Form.

A Form Design window opens.

2. Click the object you want to edit to select it.
3. Click Tools, Object Explorer.

Note

- To edit methods or procedures for the form itself, open the Object Explorer, and click the Form icon at the top of the object tree. The tabbed pages will then display the appearance, events, methods, and properties for the form.

To open the Object Explorer from a report

1. With the report open, click View, Design Report.

A Report Design window opens.

2. Click the object you want to edit.
3. Click Tools, Object Explorer.

To open the Object Explorer from a script

1. Click File, Open, Script.
2. Enable the Edit The Script button at the bottom of the Open Script dialog box.
3. Double-click the script you want to open.

The script opens in an Editor window and the built-in **run** method is displayed.

4. Click Tools, Object Explorer.

To open the Object Explorer from a library

1. Click File, Open, Library.

An empty Library Design window opens.

2. Click Tools, Object Explorer.

{button ,AL(`INTRO_EXPLORER;',0,"Defaultoverview",)} **Related Topics**

Pinning the Object Explorer to the desktop

When the Object Explorer is pinned it remains open when you move between the Form Design window and the Editor window. The Object Explorer automatically opens with the design window, and stays open until you leave the Form Design window.

To pin the Object Explorer temporarily during work on a given form

1. Open the Object Explorer.
2. Click View, Pin Explorer from the Object Explorer menu.

When you leave the Form Design window and the Editor windows associated with the form, the temporary preference is discarded.

To open the Object Explorer and keep it pinned to the desktop

1. Open the Object Explorer.
2. Click Edit, Developer Preferences.
3. Click the Explorer tab.
4. Enable the Keep Pinned check box.

{button ,AL(`INTRO_EXPLORER;',0,"Defaultoverview",)} Related Topics

About the object tree

The object tree is one of the two panes displayed within the Object Explorer. It lists the objects in your document.

Use the mouse or Arrow keys to move within the object tree. You can use the object tree to switch to a different object, without returning to the actual document. When you select a new object on the tree, the tabbed pane changes to display the appearance, methods, events, and properties for that object.

Forms

A form's object tree shows the hierarchy of objects in your document. The active object appears at the far left, and the tree that shows the container hierarchy extends to the right.

When you place an object in a form, Corel Paradox gives it a default name that begins with a pound sign (#). The object tree shows both the objects you have placed and named and the objects you have placed but not named.

Names of objects that have ObjectPAL methods attached to them are underlined and marked with an asterisk.

Reports

A report's object tree shows a diagram of the bands, fields, and design objects in your report and their relationships to one another.

{button ,AL(`INTRO_EXPLORER_TREE;INTRO_INTRO;',0,"Defaultoverview",)} Related Topics

Opening the object tree

When you open the Object Explorer from a Form Design or Report Design window, it displays two panes. One pane contains the object tree, and the other contains tabbed pages that list the object's methods, events, appearance, and properties.

To view only the object tree

- Click View, Object Tree on the Object Explorer menu.

To view both panes

- Click View, Both on the Object Explorer menu.

Note

- When you open the Object Explorer from a Library or Script window, it displays only a tabbed pane.

{button ,AL(`INTRO_EXPLORER_TREE;',0,"Defaultoverview",)} **Related Topics**

Viewing the document's structure

The object tree in the Object Explorer window displays your document's structure.

To expand and collapse the tree

- Click the plus (+) icon to expand that node of the tree.
- Click the minus (-) icon to collapse that node of the tree.

When fully expanded, the object tree shows all objects on the active form.

{button ,AL(`INTRO_EXPLORER_TREE;',0,"Defaultoverview",)} Related Topics

Copying objects

The Object Explorer allows you to copy an object from its object tree to another object.

To copy an object to another object

1. Open the Object Explorer.
2. Choose the object you want to copy from the object tree.
3. Click Edit, Copy from the Object Explorer menu.
4. Choose the container object from the object tree.
5. Click Edit, Paste.

If the container cannot accept the object, either because of containership rules or because the object is too large, you'll hear a beep and the move will not be made.

{button ,AL(`INTRO_EXPLORER_TREE;',0,"Defaultoverview",)} Related Topics

Moving objects

The Object Explorer allows you to move one object easily into another from its object tree.

To move an object into another object

1. Open the Object Explorer.
2. Choose the object you want to move from the object tree.
3. Click Edit, Cut from the Object Explorer menu.
4. Choose the container object from the object tree.
5. Click Edit, Paste.

If the container cannot accept the object, either because of containership rules or because the object is too large, you'll hear a beep and the move will not be made.

{button ,AL(`INTRO_EXPLORER_TREE;',0,"Defaultoverview",)} Related Topics

Deleting objects

The Object Explorer allows you to delete an object easily from its object tree.

To delete an object from the object tree

1. Open the Object Explorer.
2. Choose the object you want to delete from the object tree.
3. Click Edit, Delete from the Object Explorer menu.

 **Note**

- Be sure to close the object before you try to delete it. The Object Explorer won't let you delete an open object.

{button ,AL(`INTRO_EXPLORER_TREE;',0,"Defaultoverview",)} **Related Topics**

About the tabbed pane

The tabbed pane contains separate pages that list the current object's built-in or custom methods, events, and properties. This pane lets you define new custom methods and change an object's properties. The tabbed pane allows you to distinguish between methods, events, and properties of ActiveX controls and native Windows controls and form-level features.

Each item on the Methods and Events pages can be edited in its own Editor window. Double-click an item on one of these pages, or SHIFT-click several items and press ENTER to open several Editor windows at once. When you add code to an event or method, its name moves to the top of the list, and a little blue ball appears beside it to let you know you have attached custom code.

When you declare a variable, constant, or procedure in one of these windows, it is visible to all methods attached to that object.

F1 Help

Press F1 in the Object Explorer to get help on the selected item in the tabbed pane. If there is only one Help topic for the selected item, pressing F1 takes you directly to that topic. If Help contains multiple topics for the selected word, a Search dialog box lists the topics available for that language element. Select a topic and click Go To. You can also use the Alphabetical List of Methods.

{button ,AL(`INTRO_EXPLORER_PANEL;',0,"Defaultoverview",)} Related Topics

About the Appearance page

The Appearance page contains the characteristics that determine the appearance of an object. From this page, you can determine an object's color, background pattern, fonts, or frame thickness.

These characteristics can be viewed alphabetically, or by category. You can choose which way the list is organized by clicking View, Categories.

On the Appearance page, the characteristic's name is listed in the left column, and its current setting is displayed in the right column. You can change the characteristic by clicking on either its name or its setting. Some fields can be edited, so you can type in your own text; others will display a list box where you can choose other possible settings.

{button ,AL(`INTRO_EXPLORER_PANEL;',0,"Defaultoverview",)} Related Topics

About the Methods page

The Methods page list box displays all of the possible methods for the selected object, and a New Method option. Methods that have been coded appear in bold text.

Double-clicking a method in the list opens the method in an Editor window.

The Methods list contains the following items:

Use	To declare
Uses	Procedures used by the object's methods
Var	Variables
Const	Constants
Type	Types
Proc	Procedures
New method	Create a new method by double-clicking New method, entering its name in the New Method Name text box, and clicking OK. An Editor window opens for the new method.
Built-in event methods	
Custom methods	

Objects that contain custom code are marked with a small blue ball.

Notes

- The Methods page might include prototypes for methods that cannot be ovewritten in an Editor.
- You can select more than one method at a time to open multiple Editor windows. After selecting the methods, press ENTER.

{button ,AL(`INTRO_EXPLORER_PANEL;',0,"Defaultoverview",)} **Related Topics**

About the Events page

The Events page lists the events associated with the selected object, including ObjectPAL built-in event methods. Events that have been coded appear in bold text.

Double-clicking an event in the list opens the event in an Editor window.

ActiveX controls have methods, events, and properties. Their control methods are called directly, just like functions; in addition, in the case of controls, events and methods are distinct and separate. Therefore, an Events category is required. You'll find a control event on the Events page; whereas, the methods associated with it appear on the Methods page. ActiveX controls are marked with a small red ball.

In ObjectPAL, many methods can be considered to be events, or event-like, although they can be driven as if they were methods. Because the ObjectPAL built-in event methods are event-related (that is, they respond to events), they are on the Events page and are referred to in the documentation as built-in event methods.

When you add code to an event or method, its name moves to the top of the list, by default, and a little blue ball appears beside it to let you know you have attached custom code. You can change the way custom methods, ActiveX controls, and built-in event methods sort in the Developer Preferences dialog box.

{button ,AL(`INTRO_EXPLORER_PANEL;',0,"Defaultoverview",)} **Related Topics**

About the Properties page

The Properties page lists all of the properties of the selected object.

The Properties list can be viewed alphabetically or by category. You can choose which way the list is organized by clicking View, Categories.

On the Properties page, the property name is listed in the left column, and its current setting is displayed in the right column. You can edit the property by clicking either its name or its setting. Some fields can be edited, so you can type in your own text; others display a list box where you can choose other possible settings.

{button ,AL(`INTRO_EXPLORER_PANEL;',0,"Defaultoverview",)} **Related Topics**

About methods and events for ActiveX controls

ActiveX controls are embedded in Corel Paradox as form objects. Certain events, methods, and properties associated with ActiveX controls are common to all visual objects on the form. To distinguish between ActiveX-specific methods and events and those methods and events associated with visual form objects, ActiveX-specific methods and events are marked with a red ball.

{button ,AL(`INTRO_EXPLORER_PANEL;',0,"Defaultoverview",)} **Related Topics**

Viewing the tabbed pane

When you open the Object Explorer from a design window, it displays two panes. One pane contains the object tree and the other contains tabbed pages that list the object's methods, events, properties, and appearance.

To view only the tabbed tree

- Click View, Tabbed Pane on the Object Explorer menu.

To view both panes

- Click View, Both on the Object Explorer menu.

Note

- When you open the Object Explorer from a Library or Script window, it displays only the tabbed pane.

{button ,AL(`INTRO_EXPLORER_PANEL;',0,"Defaultoverview",)} **Related Topics**

Creating a new method

You can create a new method from the Object Explorer.

To create a new method

1. Open the Object Explorer.
2. Click the Methods tab.
3. Double-click New Method.
4. Type a name for the new method, and click OK.

An Editor window opens for your new method.

{button ,AL(`INTRO_EXPLORER_PANEL;',0,"Defaultoverview",)} Related Topics

Editing a method or event

You can edit an existing method or event from the Object Explorer.

To edit a method or event

1. Open the Object Explorer.
2. Click the Methods tab or the Events tab.
3. Double-click the method or event you want to edit.

An Editor window opens.

4. Type your code and save it.

{button ,AL(`INTRO_EXPLORER_PANEL;',0,"Defaultoverview",)} Related Topics

Deleting a method or event

You can delete a method or event from the Object Explorer.

To delete a method or event

1. Open the Object Explorer.
2. Click the Methods tab or the Events tab.
3. Right-click the method or event from the list provided, and choose Delete Method or Delete Event.

This selection is dimmed if this task is unavailable for the selected event or method.

{button ,AL(`INTRO_EXPLORER_PANEL;',0,"Defaultoverview",)} Related Topics

Attaching methods to a form

You can attach frequently used custom methods to a form. This is more efficient than copying the method to each object that calls the method.

You can attach methods from the Object Explorer's object tree pane.

To attach methods to a form

1. Open the Object Explorer.
2. Click on the Form icon at the top of the object tree to select the form by itself.
3. On the Methods tab do one of the following:

- choose and edit a method
- create a custom method as you would for any other object

{button ,AL(`INTRO_EXPLORER_PANEL;',0,"Defaultoverview",)} Related Topics

Changing appearance

To change an object's appearance

1. Open the Object Explorer.
2. Click the Appearance tab to display a list of the object's traits.
3. Click once on a trait to edit the field, or to see a list box with possible settings. For color or graphical settings, the arrow will display a palette or graphic list.
4. Select the new setting.

{button ,AL(`INTRO_EXPLORER_PANEL;',0,"Defaultoverview",)} Related Topics

Changing properties

To change an object's properties

1. Open the Object Explorer.
2. Click the Properties tab to display a list of the object's properties (in ObjectPAL syntax).
3. Click once on a property to edit the field, or to see a list box of choices.
4. Select the new properties.

Note

- Not all properties can be edited. A small lock icon appears to the left of read-only properties.

{button ,AL(`INTRO_EXPLORER_PANEL;',0,"Defaultoverview",)} Related Topics

About the IDE

When you work in the ObjectPAL integrated development environment (IDE), you are in either the Editor, the Debugger, the Object Explorer, or a design window.

The Editor is connected to the ObjectPAL compiler; the compiler translates the ObjectPAL code you write into machine code a computer can execute. When you use the Editor, the compiler can check your code and report any syntax errors to allow you to correct them before you try to run the application.

The Editor also works with the Debugger. These two, along with the design window and the Object Explorer, provide you with an integrated development environment.

Using the Editor you can edit

- methods (built-in or custom)
- procedures
- uses
- types
- constants
- variables

With the Debugger you can debug methods or procedures.

The code you edit or debug can be attached to a Script, a Library, to a form, or to an object in a form.

{button ,AL(`INTRO_IDE;INTRO_INTRO;',0,"Defaultoverview",)} Related Topics

Setting developer preferences

You can control many elements of the IDE by setting your preferences in the Developer Preferences dialog box.

To set developer preferences

1. Click Edit, Developer Preferences.
2. Make your changes on any of the five pages, and click OK.

{button ,AL(`INTRO_IDE;',0,"Defaultoverview",)} **Related Topics**

About the Editor

The ObjectPAL Editor is a full-featured editor that includes incremental search, smart tab indent, multiple and group undo, and many other features. It also supports BRIEF- and Epsilon-style editing. In an ObjectPAL Editor window you can write, edit, compile, and debug the ObjectPAL code that is attached to methods on a form, library, or script. The ObjectPAL Editor works the same whether you are working with an object, a form, a library, a script, or an SQL query.

There are two list boxes at the top of the Editor windows. The list box on the left allows you to move between the objects in your document. The list box on the right shows all possible methods or events for the selected object.

In addition, the Editor window is your gateway to the ObjectPAL language reference:

- You can click View, ObjectPAL Quick Lookup to display the ObjectPAL language reference for each object type, the syntax for each method, the properties and property values for each object, and the constants for things like window attributes and error codes.
- Press F1 while the cursor is in an ObjectPAL language element in the Editor, or while a language element is highlighted in the ObjectPAL Quick Lookup or the Object Explorer, to open a Help topic specific to that language element.

{button ,AL(`INTRO_IDE;',0,"Defaultoverview",)} **Related Topics**

About the Editor menus

You can press F1 when you are on a menu command to see what it does.

{button ,AL(`INTRO_IDE;',0,"Defaultoverview",)} **Related Topics**

Working in the Editor

When an Editor window opens for the first time, some default text appears. For example, if you're editing the open method, the first line reads `method open (var eventInfo Event)`, the second line is blank, and the third line reads `endMethod`. If you accidentally change the default text, you can edit it as you would any other text.

The cursor is positioned on line two to allow you to start typing right away, but you don't have to start typing on that line. You can use the mouse or Arrow keys to move the cursor and you can insert blank lines by pressing ENTER.

By default, keywords appear in bold and comments in italics. Comments in code for the ObjectPAL built-in event methods are preceded by a semicolon. You can change text attributes in the Developer Preferences dialog box.

The Editor does not automatically wrap lines of text. A line extends to the right until you press ENTER to begin a new line.

There are two list boxes at the top of the Editor window. The list box on the left allows you to move between the objects in your document. The list box on the right shows all possible methods or events for the object.

Notes

- Variables, constants, and procedures declared in a method's Editor window are visible only to that method. To make a variable, constant, or procedure visible to all of an object's methods, select Uses, Type, Const, Var, and Proc (as many of these as you want) in the Object Explorer. Corel Paradox opens a separate window for each item you choose. This is called scoping.

{button ,AL(`INTRO_IDE;',0,"Defaultoverview",)} **Related Topics**

Starting the Editor

To start the Editor from a form or an object in a form

1. With a design window open, click Tools, Object Explorer.

The Object Explorer opens and lists the methods, events, appearance, and properties associated with the object. The Methods page also includes the item's Uses, which declare external routines; and Var, Type, Const, and Proc, which declare variables, types, constants, and procedures.

A little blue ball appears before an item to indicate that it has custom code attached to it. A larger red ball indicates that the item is an ActiveX control. A lock icon indicates that the item is read-only.

2. On the Methods or Events pages, double-click the item you want to edit.

An Editor window opens.

Note

- You can select several methods at once, and simply press ENTER.

A separate Editor window opens for each item you selected. You can open as many windows as your system allows, in any order.

You must edit each method in its own window; however, you can edit more than one method at a time by opening multiple windows.

To start the Editor from a Library

1. Open the Object Explorer.
2. On the tabbed pages, double-click the item you want to edit.

An Editor window opens.

To start the Editor from a Script

1. Click File, Open, Script.
2. Enable the Edit The Script button at the bottom of the Open Script dialog box.
3. Double-click the script you want to open.

When you open a Script, it is automatically placed in an Editor window.

The first line names the method and the last line ends the method.

Note

- After you open an Editor window, you can choose other objects, methods, and events without returning to the design window. There are two list boxes at the top of the Editor window. The one on the left allows you to choose a new object within the document; the one on the right lists all of the methods and events possible for the selected object.

{button ,AL(`INTRO_IDE;',0,"Defaultoverview",)} **Related Topics**

Moving around the Editor with the keyboard

Use the following keys to move around the Editor:

CTRL + Left Arrow key	Moves the cursor one word to the left
CTRL + Right Arrow key	Moves the cursor one word to the right
HOME	Moves the cursor to the beginning of a line
END	Moves the cursor to the end of a line
CTRL + HOME	Moves the cursor to the beginning of the text
CTRL + END	Moves the cursor to the end of the text
PAGE UP	Moves one screen back
PAGE DOWN	Moves one screen forward
BACKSPACE	Deletes the character to the left of the cursor
DELETE	Deletes the character to the right of the cursor
INSERT	Has no effect because the Editor is always in insert mode. As you type, characters are pushed to the right. You cannot overwrite characters.
CTRL + C	Copies selected text to the Clipboard
CTRL + X	Copies selected text to the Clipboard and deletes it from the window
CTRL + V	Pastes text from the Clipboard into your method
TAB	Inserts a Tab character and pushes text to the right

{button ,AL(`INTRO_IDE;',0,"Defaultoverview",)} **Related Topics**

Selecting text

You can select a block of text by either dragging with the mouse, using the Arrow keys with SHIFT held down, or clicking with SHIFT held down to extend the selection.

To select a word

- Double-click it.

To select an entire line

- Click to the left of the line and drag the cursor. (The mouse is in position when the I-beam changes to an arrow.)

To select a block of text

Do one of the following:

- click and drag the mouse
- press SHIFT and use the Arrow keys
- click to indicate the starting position, and press SHIFT to extend the selection

Notes

- The keymapping you choose in the Developer Preferences dialog box also affects selection. Press SHIFT+F1 while on a blank space in an Editor window to see the keystrokes for the keymap you chose.
- When text is selected, what happens when you type a character (or paste from the Clipboard) depends on whether you checked Overwrite Blocks in the Developer Preferences dialog box.
- Double-clicking to the left of a line toggles a breakpoint and selects the line.

{button ,AL(`INTRO_IDE;',0,"Defaultoverview",)} **Related Topics**

Searching for text

You can search for words or character strings within an Editor window.

Incremental Search will find text from the cursor forward in either an Editor or a Debugger window. Find, and Find and Replace will find search for text in either direction.

To use Incremental Search

1. Open the Editor window and place the cursor.
2. Click Search, Incremental Search.
3. Type the characters you want to find.

The Editor highlights the first occurrence of the first character you type. Type another character, and the Editor highlights the first appearance of the pair of characters, and so on. The characters you type appear on the status bar.

If, as you type, you create a string that has no match in the remainder of the current Editor or Debugger window, you'll hear a beep, and the last character you typed will not appear on the status bar.

You can use the following keys with Incremental Search:

- BACKSPACE removes the last letter of the search combination and moves back to a prior match.
- CTRL + S searches for the next combination of the current search string.
- ESC stops incremental search and returns the Editor to its normal state.

Using Find, and Find and Replace

You can use these two commands from the Search menu to search for text from the cursor forward (or backward if you enable the Backwards option). The Find And Replace dialog box lets you replace the specified text with a specified value.

{button ,AL(`INTRO_IDE;',0,"Defaultoverview",)} **Related Topics**

Leaving the Editor

How you exit the Editor depends on what you want to do next.

To continue designing your form or to edit other methods

- Click Close on the Control menu of the Editor window.

To run your code immediately and see your form in action

- Click Program, Run.

Notes

- Any Editor windows that are open when you run a form will open again when you return to the design window.
- If the Prompt To Save option in the Developer Preferences dialog box is not enabled, you are not prompted to save your changes when you close an Editor window or run a form with an Editor window open. All your changes are automatically saved. Click Edit, Undo All Changes to discard changes before you close the Editor window. Changes you make in Editor windows are saved to disk when you save your form.
- If the Prompt To Save option in the ObjectPAL Preferences dialog box (Display page) is checked, a confirmation dialog box lets you save or cancel your changes when you close an Editor window or run a form with an Editor window open.

{button ,AL(`INTRO_IDE;',0,"Defaultoverview",)} **Related Topics**

About the ObjectPAL Quick Lookup

The ObjectPAL Quick Lookup is a tabbed dialog box that displays

- Types and Methods
- Objects and Properties
- Constants

From the three pages of the dialog box, you can insert type and method names, object names and properties, and constants into your code at the cursor (in the Editor). To do this, select the language element you want, and click the appropriate button in the dialog box.

All lists are in alphabetical order. When a panel has focus, you can type one letter to take you to the first item starting with that letter. If you enable the Show All check box (at the bottom of each page), that page will show you the full spectrum of the ObjectPAL language. If you disable the Show All check box, you'll see a beginner's subset of the language. This temporarily overrides the preference you set in the Developer Preferences dialog box (General page).

Types and Methods

The left side of this page displays a list of types. When you select a type, the right side lists the methods and procedures for that type. (Methods are marked by an M and procedures are marked by a P.)

A panel at the bottom displays a prototype.

Objects and Properties

The left side of this page displays a list of objects. When you select an object, the right side displays the properties for that object.

A panel at the bottom displays possible values for a selected property. If a property has only one possible value, nothing is displayed.

Constants

The Constants page displays a list of constant types on the left. When you select a type, the right side displays a list of constants relevant to that category.

Constants let you specify things like colors, mouse shape, menu attributes, and window styles.

{button ,AL(`INTRO_IDE;',0,"Defaultoverview",)} Related Topics

Using the ObjectPAL Quick Lookup

You can get to the ObjectPAL Quick Lookup dialog box from an Editor window.

To open the ObjectPAL Quick Lookup dialog box

1. Open an Editor window.
2. Click View, ObjectPAL Quick Lookup
3. Click one of the following tabs:

- Types and Methods
- Objects and Properties
- Constants

{button ,AL(`INTRO_IDE;W_EDITOR;',0,"Defaultoverview",)} Related Topics

About keywords

Keywords are basic language elements. They are reserved words in Corel Paradox and can be selected from the Program menu.

{button ,AL(`INTRO_IDE;',0,"Defaultoverview",)} **Related Topics**

Using keywords

With an Editor window open, you can quickly insert keywords into your code.

To use keywords

1. Click Program, Keywords to display the menu of keywords.
2. Choose an item from the Keywords list to insert the keyword into your code at the cursor.

{button ,AL(`INTRO_IDE;',0,"Defaultoverview",)} Related Topics

About delivering applications

When Corel Paradox delivers a form, report, script, or library, it removes the ObjectPAL source code but leaves the compiled code intact with the file. This lets others use the application but prevents them from seeing or changing your code.

The file-name extensions of delivered applications change as follows:

Application	Undelivered extension	Delivered extension
form	.FSL	.FDL
report	.RSL	.RDL
script	.SSL	.SDL
library	.LSL	.LDL

Note

- A delivered file is also protected from design modifications and cannot be opened in a design window. When you deliver a form or report, you do not need to deliver copies of the tables in its data model.

{button ,AL(`INTRO_DELIVER;INTRO_INTRO;iscripts_deliv_scrpt_proced;',0,"Defaultoverview",)}
Related Topics

Delivering an application

To deliver an application

- With the form or report open in a design window, click Format, Deliver.
- With a library or script open, click Program, Deliver.

This removes the ObjectPAL source code, but leaves the compiled code intact with the file. This allows others to use the application but prevents them from seeing or changing your file. This protects the ObjectPAL source code and the layout of any UIObjects on your form or report.

{button ,AL(`INTRO_DELIVER;',0,"Defaultoverview",)} Related Topics

About Database Expert code examples

The Corel Paradox Database Expert produces small applications that were built using ObjectPAL. You can study the code for these applications to help you write your own code for the same or similar application tasks. To access the applications, run the Database Expert. The expert places the forms, tables, reports, and other files that make up the applications into the directory you specify.

The forms in these applications are not delivered; therefore, the ObjectPAL code is available for you to cut and paste it into your own code.

Code in the applications built by the Database Expert can show you how to do these programming tasks:

- confirm the deletion of records
- place memo fields in Memo View
- open forms and reports
- base reports on queries
- use aliases
- allow for access to application help (both through the Toolbar and the keyboard)
- highlight records
- generate drop-down edit field values
- program an alphabet bar (Address Book and Contact Management applications)

For more information, see the common library and ObjectPAL's coding Standards.

{button ,AL(`INTRO_EXAMPLES;INTRO_INTRO;',0,"Defaultoverview",)} Related Topics

Coding standards

ObjectPAL code in the Database Expert applications always follows the same syntax.

Comments

All comments are formatted with a leading semicolon (;) For example,

```
;Comment here
```

To make comments stand out, some developers also add two forward slashes (//) after the semi-colon, but they are not necessary.

Constants

User-defined constants contain a lowercase portion and an all-caps portion. The lowercase text indicates the constant's data type, while the all-caps text indicates the name of the constant. ObjectPAL's predefined constants are formatted with camel caps (capital letters mixed with lowercase letters). For example,

```
strDBNAME    ;Database name as a string
DataBeginEdit    ;An ObjectPAL constant
```

Variables

All variables contain a lowercase portion and a camel caps portion. The lowercase text indicates the constant's data type, while the camel caps indicate the name of the variable. Examples:

```
strTableName ;TableName as a string
fCustomer    ;Customer form
rInvoice     ;Invoice report
libMain      ;Main library
```

{button ,AL(`INTRO_EXAMPLES;',0,"Defaultoverview",)} Related Topics

Common library

All applications built by the Database Expert share a common library. The library contains five custom methods. The custom methods and how to call them are described below.

LoadToolBar( )

Loads a common toolbar to all applications built by the Database Expert. The toolbar is a modification of the standard Toolbar and contains standard navigational buttons, plus insert record, delete record, undo, close form, and help buttons. **LoadToolBar** is called in the **setFocu**s method at the form level of all non launcher forms. Example:

```
libMain.loadToolBar()
```

DateNotes(strFieldName String)

Enters the current date and time at the top of the indicated memo on the current form and positions the cursor on the line below it. Example:

```
libMain.DateNotes(fldNotes)
```

CascadeDeletes( )

Removes the active record and any child records that depend on it. This method does not attempt to remove records from tables that are marked as read-only in the data model. The method is especially useful when you have referential integrity, which forces you to delete detail records before you can delete master records. Example:

```
libMain.CascadeDeletes()
```

SetKeyValue(strFieldName String, anyValue AnyType)

Creates an entry in a dynamic array to place the value of the active object. This method must be called before calling **GetKeyValue**.

Generally, **SetKeyValue** is called in the **removeFocus** method of the form. Example:

```
libMain.SetKeyValue("Customer", Customer_Rec_ID.Value)
```

GetKeyValue(strFieldName String) AnyType

Returns the value that was set into a dynamic array via **SetKeyValue**. This is used to help keep forms synchronized when you move between them.

Generally, **GetKeyValue** is called in the **setFocus** method of the form. Example:

```
Customer.locate("Customer", GetKeyValue("Customer Rec ID"))
```

SetKeyValue and **GetKeyValue** work together to synchronize forms. **SetKeyValue** places a key value in the library, which **GetKeyValue** can then access to locate the record referred to by **GetKeyValue**.

{button ,AL(`INTRO_EXAMPLES;',0,"Defaultoverview",)} Related Topics

Use of aliases

All forms create a common alias for the application in the **init** method that points to the directory where the form was loaded. Example:

```
var
   dynFile DynArray[] string
endVar

splitFullFileName(getFileName(), dynFile)
if (NOT addAlias(strDBNAME, "Standard", dynFile["DRIVE"] + dynFile["PATH"])) then
    errorClear()
endIf
```

Whenever a form, report, library, table, or help file is referred to, it uses this alias. This allows the application to work with any working directory. Example:

```
if (NOT libMain.open(strDBNAME + strLIBNAME)) then
    msgStop("Warning:", "Can't open " + strLIBNAME)
    close()
endIf
```

{button ,AL(`INTRO_EXAMPLES;',0,"Defaultoverview",)} **Related Topics**

Deleting records

Whenever you attempt to delete a record, a prompt is generated to ask whether you want to delete the record or not. This is generally done on the table frame or the multi-record object associated with the table. Example:

```
if  (eventInfo.id() = DataDeleteRecord) then
      if (msgQuestion("Warning", "Are you sure you want to delete the active record?") = "No")
then
            eventinfo.setErrorCode(peCannotDelete)
      endIf
endIf
```

{button ,AL(`INTRO_EXAMPLES;',0,"Defaultoverview",)} Related Topics

Accessing help

All forms can access help by pressing F1, and all non launcher forms can also access help by using the Toolbar.

F1 can be tested for in one of two methods.

action method

You can test for F1 in the **action** method on the form. Example:

```
if (eventinfo.id() = EditHelp then
   disableDefault
    if (NOT helpShowContext(strDBNAME + strHELP, 10)) then
      msgStop("Warning", "Cannot load help file")
  endIf
endIf
```

keyPhysical method

You can test for F1 in the **keyPhysical** method on the form. Example:

```
if (eventInfo.vChar() = "VK_F1")  then
    disableDefault
    if (NOT helpShowContext(strDBNAME + strHELP, 10)) then
     msgStop("Warning", "Cannot load help file")
    endIf
    disableDefault
endIf
```

To test for the help that is selected from the Toolbar, code is placed in the **menuAction** method for the form. Example:

```
if (eventInfo.id() = MenuHelpContents) then
    disableDefault
    if (NOT helpShowContext(strDBNAME + strHELP, 10)) then
       msgStop("Warning", "Unable to load Help file")
    endIf
endIf
```

strDBNAME is an alias for the directory where the application, including the Help file, resides. strHELP is the name of the Help file, and 10 is the context ID in the Help file.

{button ,AL(`INTRO_EXAMPLES;',0,"Defaultoverview",)} Related Topics

How forms are opened

Most forms in the Database Expert applications are designed to be opened only once. If the form is not open, it is opened. If it is already opened, it is brought to the top. Example:

```
if (NOT fOrder.attach("Customer Order Form")) then
    if (NOT fOrder.open(strDBNAME + strORDERFORM)) then
        msgStop("Warning", "Cannot open " + strORDERFORM)
    endIf
else
    fOrder.bringToTop()
endIf
```

Sometimes the code allows a form to be opened more than once. Example:

```
if (NOT fOrder.open(strDBNAME + strORDERFORM)) then
    msgStop("Warning", "Cannot open " + strORDERFORM)
endIf
```

{button ,AL(`INTRO_EXAMPLES;',0,"Defaultoverview",)} **Related Topics**

How reports are opened

Reports in the Database Expert applications are opened in a similar fashion to forms. Most are designed to be opened only once. What is different about them is that they open with Fit Width on (the equivalent of clicking View, Zoom, Fit Width). To accomplish this, reports are initially opened hidden, Fit Width is turned on, then the report is brought to top, which also displays it. Example:

```
if (NOT rCustomer.attach("Customer Report")) then
    if (rCustomer.open(strDBNAME + strCUSTRPT, WinstyleDefault + WinStyleHidden)) then
        rCustomer.menuAction(MenuPropertiesZoomFitWidth)
        rCustomer.bringToTop()
    else
        msgStop("Warning", "Cannot open " + strCUSTRPT)
    endIf
else
    rCustomer.bringToTop()
endIf
```

Sometimes the code allows a report to be opened more than once. Example:

```
if (rCustomer.open(strDBNAME + strCUSTRPT, WinsStyleDefault + WinStyleHidden)) then
    rCustomer.menuAction(MenuPropertiesZoomFitWidth)
    rCustomer.bringToTop()
else
    msgStop("Warning", "Cannot open " + strCUSTRPT)
endIf
```

{button ,AL(`INTRO_EXAMPLES;',0,"Defaultoverview",)} **Related Topics**

Reports based on queries

Some reports are based on queries when they are launched. To have as few fields as possible protected by the Database Expert, a check is placed under the table name. This causes all fields from the table to be included in the query.

The applications use two techniques to open reports based on queries. The first technique runs a query to a table and then opens the report based on the query. To use this technique, a report based on the query cannot already be open. If the report is already open, it is closed before the query is executed. Example:

```
if (rCustomer.attach("Customer Report")) then
    rCustomer.close()
endif
qCustomer =
    Query
        ~(strDBNAME)Customer | Customer Rec ID          |
        Check                | ~(Customer_Rec_ID.value) |
    endQuery
_recRepInfo.masterTable = ":priv:ANSWER"
_recRepInfo.name = strDBNAME + strCUSTRPT
if (rCustomer.open(_recRepInfo, WinStyleDefault + WinStyleHidden)) then
    rCustomer.menuAction(MenuPropertiesZoomFitWidth)
    rCustomer.bringToTop()
else
    msgStop("Warning", "Cannot open " + strCUSTRPT)
endIf
```

Second technique

A second technique is to open the report based on a query string. The advantage of this technique is that an open report does not have to be closed because the query-string guarantees a unique Answer table. Example:

```
strQuery =
    "Query
    " + strDBNAME + "Customer | Customer Rec ID       |
                                  Check               | " + string(Customer_Rec_ID.value) + "
|
    endQuery"
_recRepInfo.queryString = strQuery
_recRepInfo.name = strDBNAME + strCUSTRPT
if (rCustomer.open(_recRepInfo, WinstyleDefault + WinStyleHidden)) then
    rCustomer.menuAction(MenuPropertiesZoomFitWidth)
    rCustomer.bringToTop()
else
    msgStop("Warning", "Cannot open " + strCUSTRPT)
endIf
```

{button ,AL(`INTRO_EXAMPLES;',0,"Defaultoverview",)} **Related Topics**

Memo fields

All memo fields are placed in Memo View when they are selected. This is done by putting code in the **arrive** method of the memo field. Example:

```
doDefault
self.action(EditEnterMemoView)
```

{button ,AL(`INTRO_EXAMPLES;',0,"Defaultoverview",)} **Related Topics**

Record highlighting

Numerous forms, such as the Checkbook form, use record highlighting to help indicate the active record in a TableFrame.

To do record highlighting, code that sets the selected or unselected color must be placed in numerous methods. The following example shows how record highlighting is implemented in the Checkbook application. To indicate that the active record is selected, place the following code in the **canDepart** method for the table frame, and in the **arrive** method of any standalone fields:

```
recCheckbk.color = Yellow
```

Similarly, place the following code in the **arrive** method of the record object:

```
self.color = Yellow
```

To indicate that the active record is not selected, place the following code in the **canArrive** method of the table frame and in the **depart** method of any standalone fields:

```
recCheckbk color = Transparent
```

Similarly, place the following code in the **canDepart**, **depart**, and **open** methods of the record object:

```
self.color = Transparent.
```

{button ,AL(`INTRO_EXAMPLES;',0,"Defaultoverview",)} **Related Topics**

Drop-down edit fields that are built on-the-fly

Several forms, such as the Checkbook application form, contain drop-down edit fields that allow you to select from values that are contained in a specific field of a table. To do this, a query is run in the **arrive** method of the list object of the field, and the drop-down edit field is populated from the query. Example:

```
var
    tcChecksQuery    tcursor
    qbeUniqueCategory query
endVar

    qbeUniqueCategory =
        Query
            ~(strDBNAME + strCHECKBKTABLE) | Category     |
                                           | Check NOT Blank   |
        EndQuery

    ;// place the results of the query into a tcursor --> very fast!
    if (NOT qbeUniqueCategory.executeQBE(tcChecksQuery)) then
        msgInfo("Listing error" , "Failed to create Category List")
        return
    endIf
    ;// empty the list
    CategoryList.list.count = 0

    ;// set the list items pointer to the first one
    CategoryList.list.selection = 0

    ;// fill the list
    scan tcChecksQuery:
        CategoryList.list.selection = CategoryList.list.count + 1
        CategoryList.list.value = tcChecksQuery.(strCategory)
    endScan
```

{button ,AL(`INTRO_EXAMPLES;',0,"Defaultoverview",)} **Related Topics**

Alphabet bar

The Address Book application form and the Contact Management application form each have an alphabet bar that allows you to press a letter on the button bar to go to the first record that has a last name beginning with the letter pressed. If no such name is found, a prompt asks if you want to insert a new record.

The alphabet bar is a field that contains 26 buttons. The code to find last names is on the New Value method of the field.

```
var
    dynAddressFilter DynArray[] String
  lCaseFlag Logical
endVar

;// if the reason for the refresh is a value being edited on
;// the field (which in this case would be pressing on a button)
;//

    if (eventinfo.reason()= editValue) then
            doDefault
         lCaseFlag = isIgnoreCaseInLocate()
         ignoreCaseInLocate(Yes)
          if (NOT locatePattern("Last Name" , self.value + "..")) then
                if (msgQuestion("No last names starting with " + self.value, "Do you want to
add a new address?")) = "Yes" then
                mroAddressInfo.postaction(DataInsertRecord)
            endIf
       endIf
        ignoreCaseInLocate(lCaseFlag)
        ;// un-pop the button
      ;//
        self.value = ""
      First_Name.moveTo()
     endIf
```

{button ,AL(`INTRO_EXAMPLES;',0,"Defaultoverview",)} **Related Topics**

Version 5.0 ObjectPAL properties

AttachedHeader
AvgCharSize
BottomBorder (read-only)
Breakable
ByRows
CalculatedField
CheckedValue
ColumnPosition
ColumnWidth
CurrentColumn
CurrentRow
DeleteColumn
DeleteWhenEmpty
FirstRow
FrameObjects
GridLines.QueryLook
GridValue
GroupObjects
GroupRecords
Header
HeadingHeight
InsertColumn
InsertField
LeftBorder (read-only)
List.Value
Margins.Bottom
Margins.Left
Margins.Right
Margins.Top
NextTabStop
OtherBandName
PageSize
PageTiling
Picture
PositionalOrder
PrinterDocument
RefreshOption
RemoveGroupRepeats
RepeatHeader
RightBorder (read-only)
RowHeight
SeeMouseMove
Series.Marker.Size
Series.Marker.Style
SeriesName
ShowGrid
SnapToGrid
SpecialField
StandardToolbar
StartPageNumbers
SummaryModifier
TitleBoxName
TopBorder (read-only)

UncheckedValue
WideScrollBar
Width
XAxisName
XAxis.Ticks.TimeFormat
XAxis.Ticks.TimeStampFormat
Xseparation
YAxisName
YAxis.Graph_Title.Text
YAxis.Ticks.TimeFormat
YAxis.Ticks.TimeStampFormat
Yseparation
ZAxisName
ZAxis.Graph_Title.Text
ZAxis.Ticks.NumberFormat
ZAxis.Ticks.TimeFormat
ZAxis.Ticks.TimeStampFormat

{button ,AL(`REF_50;',0,"Defaultoverview",)} Related Topics

Version 5.0 ObjectPAL constants

ActionEditCommand
EditInsertObject
EditPasteLink
EditSaveCrosstab

AggModifier
CumulativeAgg
RegularAgg
UniqueAgg

DateRangeType
ByDay
ByMonth
ByQuarter
ByWeek
ByYear

FileBrowserFileType
fbDM
fbPrintStyle
fbScreenStyle
fbSQL

FrameStyle
Windows3dFrame
Windows3dGroup

MenuCommand
MenuChangedPriv
MenuChangedWork
MenuChangingPriv
MenuChangingWork
MenuFormViewData
MenuHelpToolbar
MenuHelpCoach
MenuOpenProjectView

PageTilingOption
StackPages
TileHorizontal
TileVertical

PrintColor
prnPrintColor
prnPrintMonochrome

PrintDuplex
prnHorizontal
prnSimplex
prnVertical

PrinterOrientation
prnLandscape
prnPortrait

PrinterSize

prn10x14	prnEnvC6
prn11x17	prnEnvC65

prnA3
prnA4
prnA4Small
prnA5
prnB4
prnB5
prnCSheet
prnDSheet
prnEnv9
prnEnv10
prnEnv11
prnEnv12
prnEnv14
prnEnvB4
prnEnvB5
prnEnvB6
prnEnvC3
prnEnvC4
prnEnvC5
prnEnvDL
prnEnvItaly
prnEnvMonarch
prnEnvPersonal
prnESheet
prnExecutive
prnFanfoldLegalGerman
prnFanfoldStandardGerman
prnFanfoldUS
prnFolio
prnLedger
prnLegal
prnLetter
prnLetterSmall
prnNote
prnQuarto
prnStatement
prnTabloid

PrintQuality

prnDraft
prnHigh
prnLow
prnMedium

PrintSource

prnauto
prnCassette
prnEnvelope
prnEnvManual
prnLargeCapacity
prnLargeFmt
prnLower
prnManual
prnMiddle
prnOnlyOne
prnSmallFmt
prnTractor
prnUpper

SpecialFieldType

DateField
NofFieldsField
NofPagesField
NofRecsField
PageNumField
RecordNoField
TableNameField
TimeField

UIObjectType

BandTool
PageBrkTool

{button ,AL(`REF_50;',0,"Defaultoverview",)} Related Topics

Version 7 ObjectPAL properties

New properties in version 7

CurrentPage
Enabled
InactiveColor
IncludeAllData
NumberPages
PinHorizontal
PinVertical
ProgID
ShowAllColumns
SquareTabs
TabHeight
TabsAcross
TabsOnTop

{button ,AL(`REF_7;',0,"Defaultoverview",)} Related Topics

Version 7 ObjectPAL constants

New constants in version 7
BrowserOptions used with the fileBrowser (System type) procedure.
DataTransferCharset Constants
DataTransferDelimitCode Constants
DataTransferFileType Constants
DesktopPreferenceTypes Constants
MailAddressTypes Constants used with the addAddress (Mail type) method.
RegistryKeyType Constants
RestructureOperations Constants
ToolbarBitmap Constants used with the addButton (Toolbar type) method
ToolbarButtonType Constants used with the addButton (Toolbar type) method
ToolbarClusterID Constants used with the addButton (Toolbar type) method
ToolbarState Constants used with the setState (Toolbar type) method
TrackBarStyles Constants used with native Windows TrackBar control

New MenuCommand Constants were also added to version 7.

{button ,AL(`REF_7;',0,"Defaultoverview",)} Related Topics

Version 8 ObjectPAL properties

New properties in version 8

DrillDown
Font.Script
FlatLook
HTMLAction
HTMLFormParams
HTMLMethod
Label.Font.Script
LegendBox.Font.Script
Series.Graph_Title.Font.Script
TitleBox.Graph_Title.Font.Script
TitleBox.Subtitle.Font.Script
XAxis.Graph_Title.Font.Script
XAxis.Ticks.Font.Script
YAxis.Graph_Title.Font.Script
YAxis.Ticks.Font.Script
ZAxis.Graph_Title.Font.Script
ZAxis.Ticks.Font.Script

{button ,AL(`REF_75;',0,"Defaultoverview",)} Related Topics

Version 8 ObjectPAL constants

New constants in version 8

There are two new types of constants in version 8:

- RestructureOperations used with restructure (Table type).
- MailReadOptions used with readMessage (Mail type).

New constants were also added to the following types of constants:

- MenuCommandconstants
- FileBrowserFileTypes constants

{button ,AL(`REF_75;',0,"Defaultoverview",)} Related Topics

Syntax notation

The following table displays the ObjectPAL syntax notation:

Convention	Sample	Meaning
Bold font	**beep( )**	Required element (method name or parentheses). Type the bold font convention exactly as shown. Parentheses are required, even if the method takes no arguments.
Bold italic font	***tableName***	Required element (argument). Replace with a variable, expression, or literal value.
[ ] (Square brackets)	[, ***fieldName***]	Informational element indicating an optional argument.
* (Asterisk)	[, ***fieldName***] *	Informational element indicating a repeatable argument. You choose whether to repeat this argument.
{|} (Braces and bar)	{ Yes|No }	You *must* choose one of the values separated by the vertical bar.

{button ,AL(`CONVENTIONS;',0,"Defaultoverview",)} Related Topics

Required elements

In ObjectPAL syntax, required elements are displayed in **bold** or ***bold italic*** type. In the following prototype the required elements are: the method name (**load**), the parentheses, and the argument (***formName***). The rest of the prototype consists of informational elements.

```
load ( const formName String ) Logical
```

Required element	Description
Name	The name of the method or procedure
Parentheses	Parentheses are required, even if the method or procedure takes no arguments
Argument	Unless an argument is enclosed in square bracket (which makes it optional) it must be included. An argument can be a variable, an expression, or a literal (hard-coded) value. Arguments are separated by commas.

{button ,AL(`CONVENTIONS;',0,"Defaultoverview",)} Related Topics

Informational elements

Informational elements are not essential to the ObjectPAL syntax that you type for a method or procedure. Instead, arguments describe how the method or procedure works. The following table describes ObjectPAL informational elements:

Element	Description
Square brackets	Square brackets indicate an optional argument. For example, the square brackets in the following prototype indicate that you don't have to include *formTitle* in the syntax: **attach (** [const ***formTitle*** String] **)** Logical There is one exception to this rule: when an argument is an array (or DynArray), the syntax for the argument shows square brackets following the Array (or DynArray) keyword. For instance, the following syntax indicates that **enumPrinters** takes a resizeable array as an argument: **enumPrinters (** var ***printers*** Array[ ] String **)** Logical
Keywords	Keywords displayed in normal type provide information about the arguments for a method or procedure. An argument preceded by the keyword var is passed by reference. An argument preceded by the keyword const is passed as a constant. An argument itself, without either keyword, is passed by value. The keyword that follows each argument specifies its data type (e.g., String, Number, Table, or Logical). If a method or procedure returns a value, the keyword at the end of the syntax line specifies its data type. Most ObjectPAL methods and procedures return values.
Asterisks	An asterisk (*) indicates that an argument can be repeated. For example, the following syntax indicates that **message** takes one required argument, *reqTxt*, and one or more optional arguments, represented by *optTxt*. **message (** const ***reqTxt*** String [**,** const ***optTxt*** String] * **)**

{button ,AL(`CONVENTIONS;',0,"Defaultoverview",)} **Related Topics**

ObjectPAL prototypes

Prototypes are syntax statement that are presented for each ObjectPAL method and procedure. An ObjectPAL prototype consists of required elements (displayed in **bold** or ***bold italic*** type) and informational elements (displayed in normal type).

In the following prototype, the method name (**sample**), the argument (***argOne***), and the parentheses are required. The argument ***argTwo*** is optional, and remaining code is made up of informational elements.

sample (var ***argOne*** Type [, const ***argTwo*** Type] **)** Type

In ObjectPAL code, the following statements are valid:

```
; One argument, variable x stores the return value.
x = sample(custName)


; Two arguments, the return value is not used.
sample(custName, custAddress)
```

{button ,AL(`CONVENTIONS;',0,"Defaultoverview",)} Related Topics

Alternate syntax

ObjectPAL supports an alternate syntax. The standard syntax uses dot notation to specify an object, a method name, and one or more arguments. In the following prototype, **object** is an object name or UIObject variable, **methodName** represents the name of the method, and *argument* represents one or more arguments:

object.methodName (*argument* [, *argument*])

ObjectPAL's alternate syntax does not use dot notation. Instead, it specifies the object as the first argument to the method. For example,

methodName (*object* , *argument* [, *argument*])

The following statement uses the standard ObjectPAL syntax to return a lowercase version of a string:

```
theString.lower()
```

The following statement uses the alternate syntax:

```
lower(theString)
```

For clarity and consistency, use standard syntax; however, the alternate syntax is convenient when defining the calculation for a calculated field.

{button ,AL(`CONVENTIONS;',0,"Defaultoverview",)} Related Topics

Using ObjectPAL in calculated fields

A calculated field can use any of the following elements:

- literal values
- variables declared within the scope of the calculated field, and have an assigned value.
- object properties
- basic language elements
- custom methods attached to other objects or to the field itself (you must declare a UIObject variable within the scope of the calculated field and use an attach statement to associate the variable with a UIObject).
- any method or procedure in the ObjectPAL run-time library (RTL) that returns a value (including a Logical value)
- special functions (e.g., Sum and Avg) provided specifically for use in calculated fields

Note

- ObjectPAL supports an alternate syntax that can be useful when defining a calculated field.

The following table describes these elements.

Element	Comments
5	literal value
a	literal value
x	variable (must be declared within the scope of the calculated field, and have an assigned value)
x + 5	simple expression (rules for working with variables apply)
self.Name	property (displays the field's name as a String)
theBox.Color	property (displays an integer value representing the object's color)
iif(State.Value = "CA", 0.075, 0)	basic language element **iif** (the value of the calculated field depends on the value of the State field object)
uio.objCustomMethod()	custom method attached to another object (must return a value)
tc.open("orders.db")	RTL method (displays True if the open succeeds; otherwise, it displays False. TCursor must be declared within the scope of the field)
Avg([DIVEITEM.Sale Price])	special function (operates on the Sale price field of the *Diveitem* table). The table must be in the form's data model. Quotes are not used, but spaces are allowed.
tc.cAverage("Sale Price")	RTL method (TCursor must be declared and opened previously) If a field name contains spaces, quotes are required.

{button ,AL(`CONVENTIONS;',0,"Defaultoverview",)} **Related Topics**

Derived methods

Many object types include methods derived from similar methods defined for another type. For example, the Script type includes methods derived from the Form type. The diagram below displays the methods for the Script type when scripts were introduced in Corel Paradox 5.0. In version 5.0, the Script type included eleven methods: seven methods derived from the Form type, and four methods defined specifically for the Script type. The derived methods are listed as Form methods, but the information applies equally to the Script type.

When methods are derived from other types, the ObjectPAL online Help displays information about the original method only. For example, when you request help on the **save** method, Help displays information about the **save** method defined for the Form type. The information that applies to forms also applies to scripts.

The online Help topic for each type includes information on its methods that are derived from other types.

Methods for the Script Type in version 5.0

Form		**Script**
deliver		**attach**
enumSource		**create**
enumSourceToFile		**load**
methodDelete		**run**
methodGet		
methodSet		
save		

{button ,AL(`CONVENTIONS;',0,"Defaultoverview",)} **Related Topics**

Alphabetical list of ObjectPAL types

Click on a type to view a list of its methods and procedures. Each method and procedure includes syntax, a description, and sample code.

A

AddinForm
ActionEvent
AnyType
Application
Array

B

Binary

C

Currency

D

Database
DataTransfer
Date
DateTime
DDE
DynArray

E

ErrorEvent

V-Z
ValueEvent

{button ,AL(`LISTS;CATEGORIES;',0,"Defaultoverview",)} **Related Topics**

Object type categories

Data model object types
Data types
Design object types
Display managers
Event types
System data objects

List of data model objects

Click on a type to view a list of its methods and procedures. Each method and procedure includes syntax, a description, and sample code.

Database
Query
Table
TCursor
SQL

List of system data objects

Click on a type to view a list of its methods and procedures. Each method and procedure includes syntax, a description, and sample code.

DDE
DataTransfer
FileSystem
Library
Mail
Session
Script
System
TextStream

List of data types

Click on a type to view a list of its methods and procedures. Each method and procedure includes syntax, a description, and sample code.

AnyType	DynArray	OLE
Array	Graphic	Point
Binary	Logical	Record
Currency	LongInt	SmallInt
Date	Memo	String
DateTime	Number	Time

List of design object types

Click on a type to view a list of its methods and procedures. Each method and procedure includes syntax, a description, and sample code.

Menu

Native Windows Controls

PopUpMenu

Toolbar

UIObject

Recent versions of Corel Paradox allow you to control objects that are native to the Windows operating system. Native Windows controls (NWCs) allow Corel Paradox forms to deliver information to the user in the same format as Windows.

List of display managers

Click on a type to view a list of its methods and procedures. Each method and procedure includes syntax, a description, and sample code.

AddinForm
Application
Form
Report
Script
TableView

List of event types

Click on a type to view a list of its methods and procedures. Each method and procedure includes syntax, a description, and sample code.

ActionEvent	MouseEvent
ErrorEvent	MoveEvent
Event	StatusEvent
KeyEvent	TimerEvent
MenuEvent	ValueEvent

TimerEvent type

The TimerEvent type includes methods that process information used by the timer method. Timer methods are built into each design object. Use **setTimer** (defined for the UIObject type) to specify when to send timer events to an object and then modify the object's built-in **timer** method to control the object's response when a timer goes off. Use **killTimer** (defined for the UIObject type) to turn off an object's timer.

The following example assumes that a form contains a multi-record object bound to the *Customer* table. The record container in the multi-record object is named *custRecordMRO*.

For the following example, suppose you want to give the user 60 seconds to edit a record in a data entry program. After 60 seconds, you want to alert the user. To accomplish this, the built-in **action** method for *custRecordMRO* tests every action. If the action is DataArriveRecord, the method uses **killTimer** to stops old timers and uses **setTimer** to set a new timer. When the timer goes off, a message pops up alerting the user. The following code defines a constant in the Const window for *custRecordMRO*. This code makes it easy for you to change the time:

```
; custRecordMRO::Const
const
  alertTime = 60000     ; data-entry alert at 60 seconds
endConst
```

The following code is attached to the **action** method for *custRecordMRO*;

```
; custRecordMRO::action
method action(var eventInfo ActionEvent)
if eventInfo.id() = DataArriveRecord then ; when opening to a new record
  self.killTimer()         ; just in case it hasn't expired
                           ; yet, kill the old timer
  self.setTimer(alertTime) ; start timer for this record
endif
endMethod
```

This code is attached to the **timer** method for *custRecordMRO*:

```
; custRecordMRO::timer
method timer(var eventInfo TimerEvent)
self.killTimer()
beep()
msgInfo("Alert", "You have been processing this record for " +
                 "one minute now.")
endMethod
```

The Timer type contains only derived methods from the Event type.

Methods for the TimerEvent Type

Event ⟵	**TimerEvent**
errorCode	(All TimerEvent methods are derived methods from Event type.)
getTarget	
isFirstTime	
isPreFilter	
isTargetSelf	
reason	
setErrorCod	
setReason	

Alphabetical list of 4.5 methods

beginTransaction (Database Type)

commitTransaction (Database Type)
dlgExport (System type) Moved to DataTransfer type in version 7
dlgImportASCIIFix (System type) Moved to DataTransfer type in version 7
dlgImportASCIIVar (System type) Moved to DataTransfer type in version 7
dlgImportSpreadsheet (System type) Moved to DataTransfer type in version 7
dmGetProperty (Form type)
dmLinkToFields (Form type)
dmLinkToIndex (Form type)
dmSetProperty (Form type)
dmUnlink (Form type)
enumAliasLoginInfo (Session type)
errorHasErrorCode (System type)
errorHasNativeErrorCode (System type)
errorNativeCode (System type)
executeSQL (SQL type)
executeString (System type)
exportASCIIFix (System type) Moved to DataTransfer type in version 7
exportASCIIVar (System type) Moved to DataTransfer type in version 7
exportSpreadsheet (System type) Moved to DataTransfer type in version 7
forceRefresh (TCursor type)
forceRefresh (UIObject type)
getAliasProperty (Session type)
importASCIIFix (System type) Moved to DataTransfer type in version 7
importASCIIVar (System type) Moved to DataTransfer type in version 7
importSpreadsheet (System type) Moved to DataTransfer type in version 7
isOnSQLServer (TCursor type)
isOpenOnUniqueIndex (TCursor type)
rollBackTransaction (Database type)
sendKeys (System type)
sendKeysActionID (System type)
setAliasPassword (Session type)
setAliasProperty (Session type)
setBatchOff (TCursor type)
setBatchOn (TCursor type)
TransactionActive (Database type)
update (TCursor type)
writeSQL (SQL type)

{button ,AL(`REF_45;',0,"Defaultoverview",)} **Related Topics**

Version 4.5 methods by type

The following list contains methods that were added or changed for version 4.5. Some of these are derived methods from other types.

Database type
beginTransaction
commitTransaction
rollBackTransaction
transactionActive

Form type
dmGetProperty
dmLinkToFields
dmLinkToIndex
dmSetProperty
dmUnlink

Session type
enumAliasLoginInfo
getAliasProperty
setAliasPassword
setAliasProperty

SQL type
executeSQL
writeSQL

System type
dlgExport (moved to DataTransfer Type in version 7)
dlgImportASCIIFix (moved to DataTransfer type in version 7)
dlgImportASCIIVar (moved to DataTransfer type in version 7)
dlgImportSpreadsheet (moved to DataTransfer type in version 7)
errorHasErrorCode
errorHasNativeErrorCode
errorNativeCode
executeString
exportASCIIFix (moved to DataTransfer type in version 7)
exportASCIIVar (moved to DataTransfer type in version 7)
exportSpreadsheet (moved to DataTransfer type in version 7)
importASCIIFix (moved to DataTransfer type in version 7)
importASCIIVar (moved to DataTransfer type in version 7)
importSpreadsheet (moved to DataTransfer type in version 7)
sendKeysActionID
sendKeys

TCursor type
forceRefresh
isOnSQLServer
isOpenOnUniqueIndex
setBatchOff
setBatchOn
update

UIObject type
forceRefresh

{button ,AL(`REF_45;',0,"Defaultoverview",)} **Related Topics**

Alphabetical list of version 5.0 methods

The following are lists of methods that were added or changed for version 5.0. Some of these are derived methods from other types.

New to version 5

addProjectAlias (Session type)
bringToFront (UIObject type)
canLinkFromClipboard (OLE type)
clipboardErase (Binary type)
clipboardHasFormat (Binary type)
compileInformation (System type)
copyToToolbar (UIObject type)
create (Library type)
create (Script type) Derived from Form type
createIndex (Table type)
createIndex (TCursor type)
deliver (Library type) Derived from Form type
deliver (Report type) Derived from Form type
deliver (Script type) Derived from Form type
desktopMenu (System type)
disablePreviousError (Form type)
dmAddTable (Report type) Derived from Form type
dmAttach (Form type)
dmAttach (TCursor type)
dmBuildQueryString (Form type)
dmBuildQueryString (Report type) Derived from Form type
dmEnumLinkFields (Form type)
dmEnumLinkFields (Report type) Derived from Form type
dmGetProperty (Report type) Derived from Form type
dmHasTable (Report type) Derived from Form type
dmLinkToFields (Report type) Derived from Form type
dmLinkToIndex (Report type) Derived from Form type
dmRemoveTable (Report type) Derived from Form type
dmResync (Form type)
dmSetProperty (Report type) Derived from Form type
dmUnlink (Report type) Derived from Form type
dropGenFilter (Table type)
dropGenFilter (TCursor type)
dropGenFilter (UIObject type)
enableExtendedCharacters (System type)
enumClipboardFormats (Binary type)
enumDataModel (Form type)
enumDataModel (Report type) Derived from Form type
enumFamily (Database type)
enumPrinters (System type)
enumRTLErrors (System type)
enumServerClassNames (OLE type)
enumSource (Report type) Derived from Form type
enumSource (Script type) Derived from Form type

enumSourceToFile (Report type) Derived from Form type
enumSourceToFile (Script type) Derived from Form type
enumTableLinks (Report type) Derived from Form type
exportParadoxDOS(System type) Moved to DataTransfer type
formatGetSpec (System type)
formatStringToDate (System type)
formatStringToNumber (System type)
formReturn (Script type) Derived from Form type
fromHex (AnyType type)
getDefaultPrinterStyleSheet (System type)
getDefaultScreenStyleSheet (System type)
getFileName (Report type) Derived from Form type
getGenFilter (Table type)
getGenFilter (TCursor type)
getGenFilter (UIObject type)
getIndexName (TCursor type)
getLanguageDriver (System type)
getProtoProperty (Form type)
getProtoProperty (Report type) Derived from Form type
getQueryRestartOptions (SQL type)
getRange (TCursor type)
getRange (Table type)
getRange (UIObject type)
getSelectedObjects (Form type)
getStyleSheet (Form type)
getStyleSheet (Report type) Derived from Form type
getUserLevel (System type)
hideToolbar (Form type)
insertObject (OLE type)
instantiateView (TCursor type)
isAssigned (SQL type)
isDesign (Form type)
isDesign (Report type) Derived from Form type
isLinked (OLE type)
isSQLServer (Database type)
isToolbarShowing (Form type)
isView (TCursor type)
linkFromClipboard (OLE type)
load (Library type) Derived from Form type
load (Report type) Derived from Form type
load (Script type) Derived from Form type
loadProjectAliases(Session type)
menuAction (Report type) Derived from Form type
methodDelete (Library type) Derived from Form type
methodDelete (Script type) Derived from Form type
methodGet (Library type) Derived from Form type
methodGet (Script type) Derived from Form type
methodSet (Library type) Derived from Form type
methodSet (Script type) Derived from Form type

moveTo (Report type) Derived from Form type
printerGetInfo (System type)
printerGetOptions (System type)
printerSetCurrent (System type)
printerSetOptions (System type)
projectViewerClose (System type)
projectViewerIsOpen (System type)
projectViewerOpen (System type)
readFromClipboard (Binary type)
readFromFile (Query type) Replaces Database type executeQBEFile
readFromString (Query type) Replaces Database type executeQBEString
removeProjectAlias (Session type)
run (Script type) Derived from Form type
save (Library type) Derived from Form type
save (Script type) Derived from Form type
saveProjectAliases (Session type)
saveStyleSheet (Form type)
saveStyleSheet (Report type) Derived from Form type
selectCurrentTool (Form type)
selectCurrentTool (Report type) Derived from Form type
sendToBack (UIObject type)
setDefaultPrinterStyleSheet (System type)
setDefaultScreenStyleSheet (System type)
setGenFilter (Table type)
setGenFilter) (TCursor type)
setGenFilter (UIObject type)
setMenu (Form type)
setMenu (Report type)
setPrivDir (FileSystem type)
setProtoProperty (Form type)
setProtoProperty (Report type) Derived from Form type
setQueryRestartOptions (SQL type)
setRange (Table type)
setRange (TCursor type)
setRange (UIObject type)
setSelectedObjects (Form type)
setStyleSheet (Form type)
setStyleSheet (Report type) Derived from Form type
setUserLevel (System type)
setWorkingDir (FileSystem type)
showToolbar (Form type)
toHex (AnyType type)
updateLinkNow (OLE type)
wantInMemoryTCursor (Query type)
wantInMemoryTCursor (SQL type)
windowClientHandle (Report type) Derived from Form type
writeToClipboard (Binary type)

Changed in version 5

addAlias (Session type)
attach (TCursor type)
beginTransaction (Database type)
cCount (Table type)
cCount (TCursor type)
create (Table type)
dmGet (Form type)
dmGetProperty (Form type)
dmHasTable (Form type)
dmLinkToFields (Form type)
dmLinkToIndex (Form type)
dmPut (Form type)
dmRemoveTable (Form type)
dmSetProperty (Form type)
dmUnlink (Form type)
enumAliasNames (Session type)
enumDatabaseTables (Session type)
enumDriverCapabilities (Session type)
enumFieldStruct (Table type)
enumFieldStruct (TCursor type)
enumFolder (Session type)
enumIndexStruct (Table type)
enumIndexStruct (TCursor type)
enumOpenDatabases (Session type)
enumRefIntStruct (Table type)
enumRefIntStruct (TCursor type)
enumTableLinks (Form type) (Table type)
enumSecStruct (Table type)
enumSecStruct (TCursor type)
enumUIObjectProperties (UIObject type)
enumUsers (Session type)
fieldType (Table type)
fieldType (TCursor type)
format (String type)
getQueryRestartOptions (Query type) Previously in Database type
isExecuteQBELocal (Query type) Previously in Database type
load (Form type)
load (Report type)
nRecords (TCursor type)
open (Report type)
print (Report type)
readFromFile (Query type) Replaces executeQBEFile
readFromFile (SQL type) Replaces executeSQLFile
readFromString (Query type) Replaces executeQBEString
readFromString (SQL type) Replaces executeSQLString
recNo (TCursor type)
seqNo (TCursor type)
setQueryRestartOptions (Query type) Previously in Database type
sleep (System type)

{button ,AL(`REF_50;',0,"Defaultoverview",)} **Related Topics**

Version 5.0 methods by type

The following list displays the methods that were added or changed for version 5.0. Some of these are derived methods from other types.

New to version 5.0

Anytype

fromHex
toHex

Binary

clipboardErase
clipboardHasFormat
enumClipboardFormats
readFromClipboard
writeToClipboard

Database

enumFamily
isSQLServer

FileSystem

setPrivDir
setWorkingDir

Form

disablePreviousError
dmAttach
dmBuildQueryString
dmEnumLinkFields
dmResync
enumDataModel
getProtoProperty
getSelectedObjects
getStyleSheet
hideToolbar
isDesign
isToolbarShowing
saveStyleSheet
selectCurrentTool
setMenu
setProtoProperty
setSelectedObjects
setStyleSheet
showToolbar

Library

create
deliver (derived from Form type)
load (derived from Form type)
methodDelete (derived from Form type)
methodGet (derived from Form type)

methodSet (derived from Form type)
save (derived from Form type)

OLE

canLinkFromClipboard
enumServerClassNames
insertObject
isLinked
linkFromClipboard
updateLinkNow

Query

readFromFile (replaces Database type executeQBEFile)
readFromString (replaces Database type executeQBEString)
wantInMemoryTCursor

Report (derived from Form type)

deliver
dmAddTable
dmBuildQueryString
dmEnumLinkFields
dmGetProperty
dmHasTable
dmLinkToFields
dmLinkToIndex
dmRemoveTable
dmSetProperty
dmUnlink
enumDataModel
enumSource
enumSourceToFile
enumTableLinks
getFileName
getProtoProperty
getStyleSheet
isDesign
load
menuAction
moveTo
saveStyleSheet
selectCurrentTool
setMenu (not a derived type)
setProtoProperty
setStyleSheet
windowClientHandle

Script (derived from Form type)

create
deliver
enumSource
enumSourceToFile
formReturn

load
methodDelete
methodGet
methodSet
run
save

Session

addProjectAlias
loadProjectAliases
removeProjectAlias
saveProjectAliases

SQL

isAssigned
getQueryRestartOptions
setQueryRestartOptions
wantInMemoryTCursor

System

compileInformation
desktopMenu
enableExtendedCharacters
enumPrinters
enumRTLErrors
exportParadoxDOS (moved to DataTransfer Type in version 7)
formatGetSpec
formatStringToDate
formatStringToNumber
getDefaultPrinterStyleSheet
getDefaultScreenStyleSheet
getLanguageDriver
getUserLevel
printerGetInfo
printerGetOptions
printerSetCurrent
printerSetOptions
projectViewerClose
projectViewerIsOpen
projectViewerOpen
setDefaultPrinterStyleSheet
setDefaultScreenStyleSheet
setUserLevel

Table

createIndex
dropGenFilter
getGenFilter
getRange
setGenFilter
setRange

TCursor

createIndex
dmAttach
dropGenFilter
getGenFilter
getIndexName
getRange
instantiateView
isView
setGenFilter
setRange

UIObject

bringToFront
copyToToolbar
dropGenFilter
getGenFilter
getRange
sendToBack
setGenFilter
setRange

Changed in version 5.0

Database type

beginTransaction

Form type

dmGet
dmGetProperty
dmHasTable
dmLinkToFields
dmLinkToIndex
dmPut
dmRemoveTable
dmSetProperty
dmUnlink
enumTableLinks
load

Query type

executeQBE
getQueryRestartOptions (previously in the Database type)
isExecuteQBELocal (previously in the Database type)
readFromFile (replaces executeQBEFile)
readFromString (replaces executeQBEString)
setQueryRestartOptions (previously in the Database type)
writeQBE

Report type

load
open
print

Session type

addAlias
enumAliasNames
enumDatabaseTables
enumDriverCapabilities
enumFolder
enumUsers
enumOpenDatabases

SQL type

readFromFile (replaces executeSQLFile)
readFromString (replaces executeSQLString)

String type

format

System type

sleep

Table type

cCount
create
enumFieldStruct
enumIndexStruct
enumRefIntStruct
enumSecStruct
fieldType

TCursor type

attach
cCount
enumFieldStruct
enumIndexStruct
enumRefIntStruct
enumSecStruct
fieldType
nRecords
recNo
seqNo

UIObject type

enumUIObjectProperties

{button ,AL(`REF_50;',0,"Defaultoverview",)} **Related Topics**

Alphabetical list of Version 7 methods

The following list displays the methods that were added or changed for version 7. Some of these are derived methods from other types.

New to version 7

addAddress (Mail type)
addAttachment (Mail type)
addButton (Toolbar type)
addressBook (Mail type)
addressBookTo (Mail type)
aliasName (TCursor type)
appendASCIIFix (DataTransfer type)
appendASCIIVar (DataTransfer type)
appendRow (Query type)
appendTable (Query type)
attach (OleAuto type)
attach (Toolbar type)
checkField (Query type)
checkRow (Query type)
clearCheck (Query type)
close (OleAuto type)
create (Toolbar type)
createAuxTables (Query type)
createQBEString (Query type)
createTabbed (Toolbar type)
deleteRegistryKey (System type)
dlgExport (DataTransfer type) Moved from System Type
dlgImport (DataTransfer type)
dlgImportAsciiFix (DataTransfer type) Moved from System Type
dlgImportAsciiVar (DataTransfer type) Moved from System Type
dlgImportSpreadSheet (DataTransfer type) Moved from System Type
dlgImportTable (DataTransfer type)
empty (DataTransfer type)
empty (Mail type)
empty (Toolbar type)
emptyAddresses (Mail type)
emptyAttachments (Mail type)
enumAutomationServers (OleAuto type)
enumConstants (OleAuto type)
enumConstantValues (OleAuto type)
enumControls (OleAuto type)
enumDesktopWindowHandles (System type)
enumEvents (OleAuto type)
enumExperts (System type)
enumFieldStruct (Query type)
enumMethods (OleAuto type)
enumObjects (OleAuto type)
enumProperties (OleAuto type)

enumRegistryKeys (System type)
enumRegistryValueNames (System type)
enumServerInfo (OleAuto type)
enumSourcePageList (DataTransfer type)
enumSourceRangeList (DataTransfer type)
enumWindowHandles (System type)
exportASCIIFix (DataTransfer type) Moved from System type
exportASCIIVar (DataTransfer type) Moved from System type
exportParadoxDOS (DataTransfer type) Moved from System type
exportSpreadsheet (DataTransfer type) Moved from System type
first (OleAuto type)
formatStringToDateTime (System type)
formatStringToTime (System type)
getAddress (Mail type)
getAddressCount (Mail type)
getAnswerFieldOrder (Query type)
getAnswerName (Query type)
getAnswerSortOrder (Query type)
getAppend (DataTransfer type)
getAttachment (Mail type)
getAttachmentCount (Mail type)
getCheck (Query type)
getCriteria (Query type)
getDesktopPreference (System type)
getDestCharSet (DataTransfer type)
getDestDelimitedFields (DataTransfer type)
getDestDelimiter (DataTransfer type)
getDestFieldNamesFromFirst (DataTransfer type)
getDestName (DataTransfer type)
getDestSeparator (DataTransfer type)
getDestType (DataTransfer type)
getKeyviol (DataTransfer type)
getMaxRows (Database type)
getMessage (Mail type)
getMessageType (Mail type)
getPosition (Toolbar type)
getProblems (DataTransfer type)
getRegistryValue (System type)
getRowID (Query type)
getRowNo (Query type)
getRowOp (Query type)
getSourceCharSet (DataTransfer type)
getSourceDelimitedFields (DataTransfer type)
getSourceDelimiter (DataTransfer type)
getSourceFieldNamesFromFirst (DataTransfer type)
getSourceName (DataTransfer type)
getSourceRange (DataTransfer type)
getSourceSeparator (DataTransfer type)
getSourceType (DataTransfer type)

getState (Toolbar type)
getSubject (Mail type)
getTableID (Query type)
getTableNo (Query type)
hasCriteria (Query type)
hide (Toolbar type)
importASCIIFix (DataTransfer type) Moved from System type
importASCIIVar (DataTransfer type) Moved from System type
importSpreadsheet (DataTransfer type) Moved from System type
insertRow (Query type)
insertTable (Query type)
invoke (OleAuto type)
isCompileWithDebug (Form type)
isCreateAuxTables (Query type)
isEmpty (String type)
isEmpty (Query type)
isMousePersistent (System type)
isQueryValid (Query type)
isValidDir (FileSystem type)
isValidFile (FileSystem type)
isVisible (Toolbar type)
loadDestSpec (DataTransfer type)
loadSourceSpec (DataTransfer type)
logoff (Mail type)
logoffDlg (Mail type)
logon (Mail type)
logonDlg (Mail type)
next (OleAuto type)
open (OleAuto type)
openObjectTypeInfo (OleAuto type)
openTypeInfo (OleAuto type)
readFromClipboard (Memo type)
readFromClipboard (String type)
registerControl (OleAuto type)
remove (Toolbar type)
removeButton (Toolbar type)
removeCriteria (Query type)
removeRow (Query type)
removeTable (Query type)
restructure (Table type)
runExpert (System type)
searchRegistry (System type)
send (Mail type)
sendDlg (Mail type)
setAnswerFieldOrder (Query type)
setAnswerName (Query type)
setAnswerSortOrder (Query type)
setAppend (DataTransfer type)
setCompileWithDebug (Form type)

setCriteria (Query type)
setDesktopPreference (System type)
setDest (DataTransfer type)
setDestCharSet (DataTransfer type)
setDestDelimitedFields (DataTransfer type)
setDestDelimiter (DataTransfer type)
setDestFieldNamesFromFirst (DataTransfer type)
setDestSeparator (DataTransfer type)
setIcon (Form type)
setKeyviol (DataTransfer type)
setLanguageDriver (Query type)
setMaxRows (Database type)
setMessage (Mail type)
setMessageType (Mail type)
setMouseShapeFromFile (System type)
setPosition (Toolbar type)
setProblems (DataTransfer type)
setRegistryValue (System type)
setRowOp (Query type)
setSource (DataTransfer type)
setSourceCharSet (DataTransfer type)
setSourceDelimitedFields (DataTransfer type)
setSourceDelimiter (DataTransfer type)
setSourceFieldNamesFromFirst (DataTransfer type)
setSourceRange (DataTransfer type)
setSourceSeparator (DataTransfer type)
setState (Toolbar type)
setSubject (Mail type)
shortName (FileSystem type)
show (Toolbar type)
transferData (DataTransfer type)
unAttach
unregisterControl (OleAuto type)
version (OleAuto type)
writeToClipboard (Memo type)
writeToClipboard (String type)

Changed in version 7

create (UIObject type)
date (Date type)
dlgExport (System type) Moved to DataTransfer type
dlgImportAsciiFix (System type) Moved to DataTransfer type
dlgImportAsciiVar (System type) Moved to DataTransfer type
dlgImportSpreadSheet (System type) Moved to DataTransfer type
exportASCIIFix (System type) Moved to DataTransfer type
exportASCIIVar (System type) Moved to DataTransfer type
exportParadoxDOS (System type) Moved to DataTransfer type
exportSpreadsheet (System type) Moved to DataTransfer type
importASCIIFix (System type) Moved to DataTransfer type

importASCIIVar (System type) Moved to DataTransfer type
importSpreadsheet (System type) Moved to DataTransfer type
init (Built-in type)
sysInfo (System type)

{button ,AL(`REF_7;',0,"Defaultoverview",)} **Related Topics**

Version 7 methods by type

The following list displays the methods that were added or changed for version 7. Some of these are derived methods from other types.

New to version 7

Database type

getMaxRows
setMaxRows

DataTransfer type

appendASCIIFix
appendASCIIVar
dlgExport (moved from System type)
dlgImport
dlgImportAsciiFix (moved from System type)
dlgImportAsciiVar (moved from System type)
dlgImportSpreadSheet (moved from System type)
dlgImportTable
empty
enumSourcePageList
enumSourceRangeList
exportASCIIFix (moved from System type)
exportASCIIVar (moved from System type)
exportParadoxDOS (moved from System type)
exportSpreadsheet (moved from System type)
getAppend
getDestCharSet
getDestDelimitedFields
getDestDelimiter
getDestFieldNamesFromFirst
getDestName
getDestSeparator
getDestType
getKeyviol
getProblems
getSourceCharSet
getSourceDelimitedFields
getSourceDelimiter
getSourceFieldNamesFromFirst
getSourceName
getSourceRange
getSourceSeparator
getSourceType
importASCIIFix (moved from System type)
importASCIIVar (moved from System type)
importSpreadsheet (moved from System type)
loadDestSpec
loadSourceSpec

setAppend
setDest
setDestCharSet
setDestDelimitedFields
setDestDelimiter
setDestFieldNamesFromFirst
setDestSeparator
setKeyviol
setProblems
setSource
setSourceCharSet
setSourceDelimitedFields
setSourceDelimiter
setSourceFieldNamesFromFirst
setSourceRange
setSourceSeparator
transferData

FileSystem type

isValidDir
isValidFile
shortName

Form type

isCompileWithDebug
setCompileWithDebug
setIcon

Mail type

addAddress
addAttachment
addressBook
addressBookTo
emptyAddresses
emptyAttachments
empty
getAddressCount
getAddress
getAttachmentCount
getAttachment
getMessage
getMessageType
getSubject
logoffDlg
logoff
logonDlg
logon
sendDlg
send
setMessage
setMessageType
setSubject

Memo type

readFromClipboard
writeToClipboard

OleAuto type

attach
close
enumAutomationServers
enumConstants
enumConstantValues
enumControls
enumEvents
enumMethods
enumObjects
enumProperties
enumServerInfo
first
invoke
next
open
openObjectTypeInfo
openTypeInfo
registerControl
unregisterControl
version

Query type

appendRow
appendTable
checkField
checkRow
clearCheck
createAuxTables
createQBEString
enumFieldStruct
getAnswerFieldOrder
getAnswerName
getAnswerSortOrder
getCheck
getCriteria
getRowID
getrowNo
getRowOp
getTableID
getTableNo
hasCriteria
insertRow
insertTable
isCreateAuxTables
isEmpty
isQueryValid

removeCriteria
removeRow
removeTable
setAnswerFieldOrder
setAnswerName
setAnswerSortOrder
setCriteria
setLanguageDriver
setRowOp

String type

isEmpty
readFromClipboard
writeToClipboard

System type

deleteRegistryKey
enumDesktopWindowHandles
enumExperts
enumRegistryKeys
enumRegistryValueNames
enumWindowHandles
formatStringToDateTime
formatStringToTime
getDesktopPreference
getRegistryValue
isMousePersistent
runExpert
searchRegistry
setDesktopPreference
setMouseShapeFromFile
setRegistryValue

Table type

restructure

TCursor type

aliasName
addButton
attach
create
createTabbed
empty
getPosition
getState
hide
isVisible
removeButton
remove
setPosition
setState
show

unAttach

Changed in version 7

Built-in type

init

Date type

date

System type

dlgExport (moved to DataTransfer type)
dlgImportAsciiFix(moved to DataTransfer type)
dlgImportAsciiVar (moved to DataTransfer type)
dlgImportSpreadSheet (moved to DataTransfer type)
exportASCIIFix (moved to DataTransfer type)
exportASCIIVar (moved to DataTransfer type)
exportParadoxDOS (moved to DataTransfer type)
exportSpreadsheet (moved to DataTransfer type)
importASCIIFix (moved to DataTransfer type)
importASCIIVar (moved to DataTransfer type)
importSpreadsheet (moved to DataTransfer type)
sysInfo

UIObject type

create

{button ,AL(`REF_7;',0,"Defaultoverview",)} **Related Topics**

Alphabetical list of version 8 methods

The following list displays the methods that were added or changed for version 8. Some of these are derived methods from other types.

New to version 8

attach (AddinForm type)
bringToTop (AddinForm type)
close (AddinForm type)
closeQuery (AddinForm type)
enumForms (AddinForm type)
enumInbox (Mail type)
fileBrowserEx (System type)
getHTMLTemplate (UIObject type)
getPosition (AddinForm type)
getPropertyAsInteger (AddinForm type)
getPropertyAsNumber (AddinForm type)
getPropertyAsString (AddinForm type)
getSender (Mail type)
getTitle (AddinForm type)
handle (TCursor type)
hide (AddinForm type)
isAppBarVisible (Toolbar type)
isAssigned (AddinForm type)
isMaximized (AddinForm type)
isMinimized (AddinForm type)
isVisible (AddinForm type)
maximize (AddinForm type)
menuAction (AddinForm type)
methodEdit (Form type)
methodEdit (Library type)
methodEdit (Script type)
methodEdit (UIObject type)
minimize (AddinForm type)
open (AddinForm type)
postMessage (AddinForm type)
readFromRTFFile (Memo type)
readMessage (Mail type)
restructure (Table type)
searchEx (String type)
sendMessage (AddinForm type)
setPosition (AddinForm type)
setProperty (AddinForm type)
setTitle (AddinForm type)
show (AddinForm type)
showApplicationBar (Toolbar type)
sizeEx (String type)
wait (AddinForm type)
writeToRTFFile (Memo type)

Changed in version 8

addButton (Toolbar type)
attach (TCursor type)
enumIndexStruct (Table type)
fill (String type)
getAddress (Mail type)
space (String type)
subStr (String type)

{button ,AL(`REF_75;',0,"Defaultoverview",)} **Related Topics**

Version 8 methods by type

The following list displays the methods that were added or changed for version 8. Some of these are derived methods from other types.

New to version 8

AddinForm type

attach
bringToTop
close
closeQuery
enumForms
getPosition
getPropertyAsInteger
getPropertyAsNumber
getPropertyAsString
getTitle
hide
isAssigned
isMaximized
isMinimized
isVisible
maximize
menuAction
minimize
open
postMessage
sendMessage
setPosition
setProperty
setTitle
show
wait
windowHandle

Form type

methodEdit

Library type

methodEdit

Mail type

enumInbox
getSender
readMessage

Memo type

readFromRTFFile
writeToRTFFile

Script type

methodEdit

String type
searchEx
sizeEx

System type
fileBrowserEx

Table type
restructure

TCursor type
handle

Toolbar type
isAppBarVisible
showApplicationBar

UIObject type
methodEdit

Changed in version 8

Mail type
getAddress

Table type
enumIndexStruct

TCursor type
attach

Toolbar type
addButton

String type
fill
space
subStr

{button ,AL(`REF_75;',0,"Defaultoverview",)} **Related Topics**

Types of constants

Click any of the following types of constants for more information:

ActionClasses constants
ActionDataCommands constants
ActionEditCommands constants
ActionFieldCommands constants
ActionMoveCommands constants
ActionSelectCommands constants
AggModifiers constants
BrowserOptions constants
ButtonStyles constants
ButtonTypes constants
Colors constants
CompleteDisplay constants
DataTransferCharset constants
DataTransferDelimitCode constants
DataTransferFileType constants
DateRangeTypes constants
DesktopPreferenceTypes constants
ErrorReasons constants
EventErrorCodes constants
ExecuteOptions constants
FieldDisplayTypes constants
FileBrowserFileTypes constants
FontAttributes constants
FrameStyles constants
General constants
GraphBindTypes constants
GraphicMagnification constants
GraphLabelFormats constants
GraphLegendPosition constants
GraphMarker constants
GraphTypeOverRide constants
GraphType constants
IdRange constants
Keyboard constants
KeyBoardStates constants
LibraryScope constants
LineEnd constants
LineStyles constants
LineThickness constants
LineTypes constants
MailAddressTypes constants
MailReadOptions constants
MenuChoiceAttributes constants
MenuCommands constants
MenuReasons constants
MouseShapes constants
MoveReasons constants
PageTilingOptions constants
PatternStyles constants
PrintColor constants
PrintDuplex constants
PrinterOrientation constants

PrinterSizes constants
PrintQuality constants
PrintSources constants
QueryRestartOptions constants
RasterOperations constants
RegistryKeyType constants
ReportOrientation constants
ReportPrintPanel constants
ReportPrintRestart constants
RestructureOperations constants
SpecialFieldTypes constants
StatusReasons constants
TableFrameStyles constants
TextAlignment constants
TextDesignSizing constants
TextSpacing constants
ToolbarBitmap constants
ToolbarButtonType constants
ToolbarClusterID constants
ToolbarState constants
TrackBarStyles constants
UIObjectTypes constants
ValueReasons constants
WindowStyles constants

ActionClasses constants

Constant	Data type	Description
DataAction	SmallInt	Data actions are for navigating in a table and for tasks such as record locking and record posting.
EditAction	SmallInt	Edit actions alter data within a field.
FieldAction	SmallInt	Field actions are a special category of Move action that enable movement between field objects.
MoveAction	SmallInt	Move actions are for moving within a field object.
SelectAction	SmallInt	Select actions are equivalent to Move actions.

{button ,AL(`OPAL_CONST_CONSTYPES;',0,"Defaultoverview",)} **Related Topics**

ActionDataCommands constants

Constant	Data type	Description
DataArriveRecord	SmallInt	Indicates a change to the active record (e.g., navigation, editing, network refresh, and scrolling)
DataBegin	SmallInt	Moves to the first record in the table associated with the given UIObject. This constant forces recursive action (DataUnlockRecord) if the active record has been modified. If an error is encountered, it calls the **error** method. This constant is invoked by the First Record button, or Record, First.
DataBeginEdit	SmallInt	Used to enter Edit mode on the form. This constant is invoked by F9, the Edit icon, or View, Edit Data.
DataBeginFirstField	SmallInt	Moves to the first field in the first record of the table associated with the given UIObject. This constant is invoked by CTRL + HOME.
DataCancelRecord	SmallInt	Discards changes to a record. Succeeds by default, but user could block it. This constant is invoked by Edit, Undo, ALT + BACKSPACE, or Record, Cancel Changes menu item. Also used internally when moving off a locked but unmodified record.
DataDeleteRecord	SmallInt	Deletes the active record. If an error is encountered, this constant calls the **error** method. This action is irreversible except for dBASE tables. This constant is invoked by Record, Delete or CTRL + DELETE.
DataDesign	SmallInt	Switches from running the form to the Form Design window. This constant is invoked by F8.
DataDitto	SmallInt	Copies into the current field the value of the corresponding field in the prior record. This constant is invoked by CTRL + D.
DataEnd	SmallInt	Moves to the final record in the table associated with the given UIObject. DataEnd forces a recursive action (DataUnlockRecord) if the active record has been modified. If an error is encountered, this constant will call the **error** method. This constant is invoked by the Last Record button.
DataEndEdit	SmallInt	Exits Edit mode on the form. This constant is invoked by (2nd) F9, Edit Data button on Toolbar, or View, View Data.
DataEndLastField	SmallInt	Moves to the last field of the last record of the table associated with a UIObject. This constant is invoked by CTRL + HOME.
DataFastBackward	SmallInt	Moves backward one set of records (where a set is defined as the number of rows in a table frame or MRO). This constant is invoked by Record, Previous Set, SHIFT + F11 or Previous Record Set button.
DataFastForward	SmallInt	Moves forward one set of records (where a set is defined as the number of rows in a table frame or MRO). This constant is invoked by Record, Next Set, SHIFT + F11 or Next Record Set button.
DataHideDeleted	SmallInt	Alters the mode of the form so that deleted records will be hidden (available only for dBASE tables). This constant is invoked by deselecting View, Show Deleted.
DataInsertRecord	SmallInt	Inserts a new (blank) record before the active record. The new record has a locked record state, and does not exist in the underlying table until the record is eventually modified and unlocked. This constant is invoked by Record, Insert, or INSERT. Note that records created this way can be discarded by using DataDeleteRecord or DataCancelRecord before they have been unlocked. If you Move off such a record without making changes, DataCancelRecord to discard it.
DataLockRecord	SmallInt	Locks the active record. If an error is encountered, this constant calls the **error** method. This constant is invoked by F5.
DataLookup	SmallInt	Invokes lookup table for the current field, to accept user's choice of a new value, and, if appropriate, to update all

		corresponding fields governed by lookup. DataLookup is available only for fields that have been defined as lookup fields. This constant is invoked by CTRL + SPACEBAR.
DataLookupMove	SmallInt	Allows the user to choose a new master record for this detail. This constant is invoked by Record, Move Help or CTRL + SHIFT + SPACEBAR.
DataNextRecord	SmallInt	Moves to the next sequential record in the table associated with the UIObject. DataNextRecord forces a recursive action (DataUnlockRecord) if the active record has been modified. If an error is encountered, this constant calls the **error** method. This constant is invoked by Record, Next, the Next Record button, F12, and so forth.
DataNextSet	SmallInt	Moves forward one set of records (where a set is defined as the number of rows in a table frame or MRO. This constant is invoked by PAGEDOWN.
DataPostRecord	SmallInt	Posts a record. DataPostRecord is just like DataUnlockRecord, but the record lock will not be released. As a consequence, if changes to key fields mean the record will move to a new position in the table, the table's position flies with that record (meaning it will still be the active record). This constant is invoked by CTRL + F5 or Record, Post/Keep Locked.
DataPrint	SmallInt	Prints a Form or Table window. This constant is invoked by File, Print or the Print button.
DataPriorRecord	SmallInt	Moves (if possible) to the previous record in the table associated with the UIObject. DataPriorRecord forces recursive action(DataUnlockRecord) if the active record has been modified. If an error is encountered, this constant calls the **error** method. This constant is invoked by Record, Previous, the Prior Record button, F11, and so forth.
DataPriorSet	SmallInt	Moves backward one set of records (where a set is defined as the number of rows in a table frame or MRO, or 1 in the case of a single-record form). DataPriorSet forces a recursive action (DataUnlockRecord) if the active record has been modified. If an error is encountered, this constant calls the **error** method. This constant is invoked by PAGEUP.
DataRecalc	SmallInt	Forces an object and all objects it contains to refetch and recompute all their data. This constant is invoked by CTRL + F3.
DataRefresh	SmallInt	Notifies users about a refresh of a value in a record displayed on the screen.
DataRefreshOutside	SmallInt	Notifies users about a refresh of a value in a record not displayed on the screen.
DataSaveCrosstab	SmallInt	Writes given crosstab to CROSSTAB.DB. Different from EditSaveCrosstab, which brings up a dialog box asking user the name of the crosstab table to create.
DataSearch	SmallInt	Opens a dialog box to allow the user to search for a specific value within a specified field. This constant is invoked by Record, Locate, Value, or CTRL + Z.
DataSearchNext	SmallInt	Searches for the next record containing the value last specified in response to the last DataSearch action. This constant is invoked by Record, Locate Next, or CTRL + A.
DataSearchRecord	SmallInt	Opens a dialog box to allow the user to search for a record by specifying the record number. This constant is invoked by Record, Locate, Record Number.
DataSearchReplace	SmallInt	Opens a dialog box to allow the user to search for a specific value within a specified field and to replace it with a different value. This constant is invoked by Record, Locate, and Replace, or CTRL + SHIFT + Z.
DataShowDeleted	SmallInt	Alters the mode of the form so that deleted records will be shown (available only for dBASE tables). They will look no different from normal records, but the status line will reflect their state. This constant is invoked by View, Show Deleted.

DataTableView	SmallInt	Opens a Table View of the master table of a form. If this form was originally invoked as preferred form of existing Table View, this returns focus to that Table View. This constant is invoked by F7, the Table View button or View, Table View.
DataToggleDeleted	SmallInt	Reverses the state of show deleted records for dBASE tables.
DataToggleDeleteRecord	SmallInt	Reverses the deleted state of records in dBASE tables.
DataToggleEdit	SmallInt	Reverses the Edit state of the form. DataToggleEdit recursively calls DataBeginEdit or DataEndEdit as appropriate. This constant is invoked by F9, or the Edit Data button.
DataToggleLockRecord	SmallInt	Reverses the lock state of the active record. DataToggleLockRecord recursively uses DataLockRecord or DataUnlockRecord as appropriate. If an error is encountered, this constant calls the **error** method.
DataUnDeleteRecord	SmallInt	Marks previously deleted record as undeleted (for dBASE tables)
DataUnlockRecord	SmallInt	Commits the record modifications to the table and then (if successful) to unlock the record. If an error is encountered, this constant calls the **error** method. This constant is invoked by Record, Unlock or SHIFT + F5.

{button ,AL(`OPAL_CONST_CONSTYPES;opal_type_actionevent;;;;;',0,"Defaultoverview",)} Related Topics

ActionEditCommands constants

Constant	Data type	Description
EditCommitField	SmallInt	Writes the current field's modifications to record buffer (without leaving field)
EditCopySelection	SmallInt	Copies selected area of text to Clipboard. This constant is invoked by Edit, Copy or CTRL + Ins.
EditCopyToFile	SmallInt	Invokes a dialog box to copy selection to a file. This constant is invoked by Edit, Copy To.
EditCutSelection	SmallInt	Copies selected area of text to Clipboard and deletes it. This constant is invoked by Edit, Cut or CTRL + DELETE.
EditDeleteBeginLine	SmallInt	Deletes from the current position to the beginning of the line
EditDeleteEndLine	SmallInt	Deletes from the current position to the end of the line
EditDeleteLeft	SmallInt	Deletes one character position to the left. This constant is invoked by Backspace in Field View.
EditDeleteLeftWord	SmallInt	Deletes up to and including the beginning of the word to the left of the current character position
EditDeleteLine	SmallInt	Deletes the line on which the current position is found
EditDeleteRight	SmallInt	Deletes one character position to the right. This constant is invoked by Del in Field View.
EditDeleteRightWord	SmallInt	Deletes up to and including the end of the word to the right of the current character position
EditDeleteSelection	SmallInt	Deletes the currently selected area of text. This constant is invoked by Edit, Delete.
EditDeleteWord	SmallInt	Deletes the word around the current position. This constant is invoked by CTRL + BACKSPACE.
EditDropDownList	SmallInt	Drops down the pick list associated with a drop-down edit field. This constant is invoked by ALT + the Down Arrow key or clicking edit field's List icon.
EditEnterFieldView	SmallInt	Enters Field View for the current field (allowing arrow keys to move around within the field). Begins by moving the current position to the end of field and unhighlighting it. This constant is invoked by F2, View, Field View, or the Field View button.
EditEnterMemoView	SmallInt	Enters Memo View on memos or OLE fields. This constant is invoked by SHIFT + F2 or View, Memo View.
EditEnterPersistFieldView	SmallInt	Enters Persistent Field View, meaning arrow keys always move within character positions within a field, even when moving to new fields. This constant is invoked by CTRL + F2 or View, Persistent Field View.
EditExitFieldView	SmallInt	Exits Field View (meaning the arrow keys will move between fields again) and highlights entire field. This constant is invoked by F2, View, Field View, or the Field View button.
EditExitMemoView	SmallInt	Exits Memo View on memos or OLE fields, meaning Enter and TAB will once again move between fields. This constant is invoked by SHIFT + F2 or View, Memo View.
EditExitPersistField View	SmallInt	Exits Persistent Field View, meaning arrow keys move between fields. This constant is invoked by CTRL + F2 or View, Persistent Field View.
EditHelp	SmallInt	Invokes the Help subsystem. This constant is invoked by F1.
EditInsertBlank	SmallInt	Inserts a blank character at the current position
EditInsertLine	SmallInt	Inserts a blank line at the current position
EditInsertObject (5.0)	SmallInt	Inserts a linked or embedded object into the current field (used only by OLE fields)
EditLaunchServer	SmallInt	Invokes the server application appropriate for the current field (used only by OLE fields)
EditPaste	SmallInt	Pastes from the Clipboard to the current position (replacing the active selection if appropriate). This constant is invoked by

		SHIFT + INSERT or Edit, Paste.
EditPasteFromFile	SmallInt	Invokes a dialog box, allowing user to select file to insert at the current position. This constant is invoked by Edit, Paste From.
EditPasteLink (5.0)	SmallInt	Pastes an object from the Clipboard and establishes a link to the underlying file (used only by OLE fields). This constant is invoked by Edit, Paste Link.
EditProperties	SmallInt	Invokes the property inspection menu for the given object. Only unbound field objects, bound graphic fields, and bound formatted memo fields support this. This constant is invoked by mouse right-click, Properties, Current Object, or F6.
EditReplace	SmallInt	Toggles overstrike mode in a field object
EditSaveCrosstab (5.0)	SmallInt	Invokes a dialog box to allow user to save a crosstab. This constant is invoked by Edit, Save Crosstab.
EditTextSearch	SmallInt	Invokes a dialog box to allow user to search and replace text within the current field. This constant is invoked by Edit, Search Text.
EditToggleFieldView	SmallInt	Reverses the current state of Field View. EditToggleFieldView recursively calls EditEnterFieldView or EditExitFieldView. This constant is invoked by F2, the Field View button, or Edit, Field View.
EditUndoField	SmallInt	Discards the current field's modifications and reverts to value in the active record buffer. This constant is invoked by ESC.

{button ,AL(`OPAL_CONST_CONSTYPES;opal_type_actionevent;;;;;',0,"Defaultoverview",)} Related Topics

ActionFieldCommands constants

Constant	Data type	Description
FieldBackward	SmallInt	Moves one field backward in tab order. This will search for the prior UIObject marked as a Tab Stop in left-right/top-down order. This constant is invoked by SHIFT + TAB.
FieldDown	SmallInt	Moves to field below the current field, whether in Field View or not. This constant is invoked by ALT + the Down Arrow key.
FieldEnter	SmallInt	Used to commit modifications to a field (if any) and to move one field forward in tab order. This constant is invoked by ENTER.
FieldFirst	SmallInt	Moves to the first field within a record. This constant is invoked by ALT + HOME.
FieldForward	SmallInt	Moves one field forward in tab order. This will search for the next UIObject marked as a Tab Stop in left-right/top-down order. This constant is invoked by Tab.
FieldGroupBackward	SmallInt	Moves one super tab group backward (e.g., between different table frames on the same form). This constant is invoked by F3.
FieldGroupForward	SmallInt	Moves one super tab group forward (e.g., between different table frames on the same form). This constant is invoked by F4.
FieldLast	SmallInt	Moves to the last field within a record. This constant is invoked by ALT + END or by END (when not in Field View).
FieldLeft	SmallInt	Moves to the field left of the current field. This constant is invoked by ALT + the Down Arrow key.
FieldNextPage	SmallInt	Moves to the next sequential page in multi-page form. This constant is invoked by View, Page, Next or SHIFT + F4.
FieldPriorPage	SmallInt	Moves to the prior page in multi-page form. This constant is invoked by View, Page, Previous or SHIFT + F3.
FieldRight	SmallInt	Moves to the field right of the current field, whether in Field View or not. This constant is invoked by ALT + the Right Arrow key.
FieldRotate	SmallInt	Used to rotate columns within a table frame. This constant is invoked by CTRL + R.
FieldUp	SmallInt	Moves to the field above the current field, whether in Field View or not. This constant is invoked by ALT + the Up Arrow key.

{button ,AL(`OPAL_CONST_CONSTYPES;opal_type_actionevent;;;;;',0,"Defaultoverview",)} Related Topics

ActionMoveCommands constants

Constant	Data type	Description
MoveBegin	SmallInt	Moves to the beginning of the document in Memo view; otherwise, it moves to the first field in the first record of table. This constant is invoked by CTRL + HOME.
MoveBeginLine	SmallInt	Moves to the beginning of the line in Memo view; otherwise, it moves to the first field in the record. This constant is invoked by HOME.
MoveBottom	SmallInt	Moves to the bottom line of the text region in Memo view. Otherwise, it moves to the last record in table.
MoveBottomLeft	SmallInt	Moves to the beginning of the last line on screen in Memo view
MoveBottomRight	SmallInt	Moves to the end of the last line on screen in Memo view. This constant is invoked by CTRL + PAGEDOWN.
MoveDown	SmallInt	Moves down as appropriate. In Memo View, it moves down one line on multi-line fields. Otherwise, it moves to the next Tab Stop object below the active object. Table frame objects move to the next record. This constant is invoked by the Down Arrow key.
MoveEnd	SmallInt	Moves to the end of the document in Memo view; otherwise, it moves to the last field in the last record of table. This constant is invoked by CTRL + END.
MoveEndLine	SmallInt	Moves to the end of the line in Memo view; otherwise, it moves to the last field in the record. This constant is invoked by END.
MoveLeft	SmallInt	Moves left as appropriate. In Memo View, it moves one character position left; otherwise, it moves to the next Tab Stop object right of the active object. This constant is invoked by the Left Arrow key.
MoveLeftWord	SmallInt	Moves the cursor to the beginning of the word to the left of the current insertion point in Memo view. This constant is invoked by CTRL + the Left Arrow key.
MoveRight	SmallInt	Moves right as appropriate. In Memo View, it moves one character position right; otherwise, it moves to the next Tab Stop object right of the active object. This constant is invoked by the Right Arrow key.
MoveRightWord	SmallInt	Moves the cursor to the beginning of the word to the right of the current insertion point. This constant is invoked by CTRL + the Right Arrow key.
MoveScrollDown	SmallInt	Scrolls the image down (effectively moving viewing area up) by appropriate amount. Active fields scroll by even lines of text. Tables move to a new record. In Memo View, scroll toward the bottom of the text. The cursor remains on the same line of the display region unless the last line of the text is visible, in which case the cursor moves down one line until the last line is reached. This constant is invoked by CTRL + the Down Arrow key.
MoveScrollLeft	SmallInt	Scrolls the image right (effectively moving viewing area to the right) by appropriate amount. Active fields scroll roughly one character position. Tables move to a new column.
MoveScrollPageDown	SmallInt	Scrolls the image down (effectively moving viewing area up) by the logical size of the object (e.g., the complete page of the document). This constant is invoked by PAGEDOWN.
MoveScrollPageLeft	SmallInt	Scrolls the image left (effectively moving viewing area right) by the logical size of the object (e.g., the complete page of the document).
MoveScrollPageRight	SmallInt	Scrolls the image right (effectively moving viewing area left) by the logical size of the object (e.g., the complete page of the document).
MoveScrollPageUp	SmallInt	Scrolls the image up (effectively moving viewing area down) by the logical size of the object (e.g., the complete page of the document). This constant is invoked by PAGEUP.
MoveScrollRight	SmallInt	Scrolls the image right (effectively moving viewing area to the

		left) by appropriate amount. Active fields scroll roughly one character position. Tables move to a new column.
MoveScrollScreenDown	SmallInt	Scrolls the image down (effectively moving viewing area up) by the size of viewing area (e.g., the size of the field). In Memo View, moves down in the document by the height of the display area.
MoveScrollScreenLeft	SmallInt	Scrolls the image left (effectively moving viewing area right) by the size of viewing area (e.g., the size of the field).
MoveScrollScreenRight	SmallInt	Scrolls the image right (effectively moving viewing area left) by the size of viewing area (e.g., the size of the field).
MoveScrollScreenUp	SmallInt	Scrolls the image up (effectively moving viewing area down) by the size of viewing area (e.g., the size of the field). In Memo View, moves up in the document by the height of the display area.
MoveScrollUp	SmallInt	Scroll the image up (effectively moving viewing area down) by appropriate amount. Active fields scroll by even lines of text. In Memo View, scroll toward the top of the document by one line of text. The cursor stays at the same line position unless the top line of the document is visible, in which case the cursor moves up one line if it can. This constant is invoked by CTRL + the Up Arrow key.
MoveTop	SmallInt	Moves the cursor to the first line of text visible in the display region in Memo view; otherwise, it moves to the first record in table.
MoveTopLeft	SmallInt	Moves to the top left of the display region in Memo view; otherwise, it moves to top left field. This constant is invoked by CTRL + PAGEUP.
MoveTopRight	SmallInt	Moves to the top right of the display region in Memo view; otherwise, it moves to top right field.
MoveUp	SmallInt	Moves up as appropriate. In Memo View, it moves up one line on multi-line fields; otherwise, it moves to the next Tab Stop object above the active object. Table frame objects move to the prior record. This constant is invoked by the Up Arrow key.

{button ,AL(`OPAL_CONST_CONSTYPES;opal_type_actionevent;;;;;',0,"Defaultoverview",)} Related Topics

ActionSelectCommands constants

Constant	Data type	Description
SelectBegin	SmallInt	In Memo View, it selects from the current position to the beginning of the document; otherwise, it selects from the current position to the first field in the first record of table. This constant is invoked by SHIFT + CTRL + HOME.
SelectBeginLine	SmallInt	In Memo View, it selects from the current position to the beginning of the line; otherwise, it selects from the current position to the first field in the record. This constant is invoked by SHIFT + HOME.
SelectBottom	SmallInt	In Field View and Memo View, select from the current position to bottom of the display region; otherwise, it selects from the current position to the last record in table.
SelectBottomLeft	SmallInt	In Memo View, it selects from the current position to the beginning of the last line in the display region. This constant is invoked by SHIFT + CTRL + PAGEUP.
SelectBottomRight	SmallInt	In Memo View, it selects from the current position to the end of the last line in the display region. This constant is invoked by SHIFT + CTRL + PAGEDOWN.
SelectDown	SmallInt	Selects down as appropriate. In Field View or Memo View, it selects down one line on multi-line fields. Cannot extend selection across fields in forms. Table frame objects select to the next record. This constant is invoked by SHIFT + the Down Arrow key.
SelectEnd	SmallInt	In Field View or Memo View, it selects from the current position to the end of the document; otherwise, it selects from the current position to the last field in the last record of table. This constant is invoked by SHIFT + CTRL + END.
SelectEndLine	SmallInt	In Field View or Memo View, it selects from the current position to the end of the line; otherwise, it selects from the current position to the last field in the record. This constant is invoked by SHIFT + END.
SelectLeft	SmallInt	Selects left as appropriate. In Field View or Memo View, it selects one character position left; otherwise, it selects the next Tab Stop object right of the active object. This constant is invoked by SHIFT + the Left Arrow key.
SelectLeftWord	SmallInt	In Field View or Memo View, if the cursor is between words, it selects the word to the left of the cursor. If the cursor is within a word, it selects to the beginning of that word. This constant is invoked by SHIFT + CTRL + the Left Arrow key.
SelectRight	SmallInt	Selects right as appropriate. In Field View or Memo View, it selects one character position right. This constant is invoked by SHIFT + the Right Arrow key.
SelectRightWord	SmallInt	In Field View or Memo View, it selects to the beginning of the next word. If the cursor precedes one or more spaces or tabs, selection only includes those spaces or tabs. This constant is invoked by SHIFT + CTRL + the Right Arrow key.
SelectScrollDown	SmallInt	Selects the image down (effectively moving viewing area up) by appropriate amount. Active fields select even lines of text. Tables select a new record. This constant is invoked by SHIFT + CTRL + the Down Arrow key.
SelectScrollLeft	SmallInt	Selects the image on left (effectively moving viewing area to the right) by appropriate amount. Active fields select roughly one character position. Tables select to a new column.
SelectScrollPageDown	SmallInt	Selects the image down (effectively moving viewing area up) by the logical size of the object (e.g., the complete page of the document).
SelectScrollPageLeft	SmallInt	Selects the image left (effectively moving viewing area right) by the logical size of the object (e.g., the complete page of the document).

SelectScrollPageRight	SmallInt	Selects the image right (effectively moving viewing area left) by the logical size of the object (e.g., the complete page of the document).
SelectScrollPageUp	SmallInt	Selects the image up (effectively moving viewing area down) by the logical size of the object (e.g., the complete page of the document).
SelectScrollRight	SmallInt	Selects the image on right (effectively moving viewing area to the left) by appropriate amount. Active fields select roughly one character position. Tables select a new column.
SelectScrollScreenDown	SmallInt	Selects the image down (effectively moving viewing area up) by the size of viewing area (e.g., the size of the field). This constant is invoked by SHIFT + PAGEDOWN.
SelectScrollScreenLeft	SmallInt	Selects the image left (effectively moving viewing area right) by the size of viewing area (e.g., the size of the field).
SelectScrollScreenRight	SmallInt	Selects the image right (effectively moving viewing area left) by the size of viewing area (e.g., the size of the field).
SelectScrollScreenUp	SmallInt	Selects the image up (effectively moving viewing area down) by the size of viewing area (e.g., the size of the field). This constant is invoked by SHIFT + PAGEUP.
SelectScrollUp	SmallInt	Moves the image up (effectively moving viewing area down) by appropriate amount. Active fields move by even lines of text.
SelectSelectAll	SmallInt	Selects the entire field
SelectTop	SmallInt	In Field View or Memo View, it selects from the current position to the top of the display region; otherwise, it selects from the current position to the first record in table.
SelectTopLeft	SmallInt	In Field View or Memo View, it selects from the current position to the beginning of screen; otherwise, it selects from the current position to the top left field. This constant is invoked by SHIFT + CTRL + PAGEUP.
SelectTopRight	SmallInt	In Field View or Memo View, it selects from the current position to the end of the top line of the screen; otherwise, it selects from the current position to the top right field. This constant is invoked by SHIFT + CTRL + PAGEDOWN.
SelectUp	SmallInt	Selects up as appropriate. In Field View or Memo View, it selects up one line on multi-line fields; otherwise, it selects the next Tab Stop object above the active object. Table frame objects select to the prior record. This constant is invoked by SHIFT + the Up Arrow key.

{button ,AL(`OPAL_CONST_CONSTYPES;opal_type_actionevent;;;;;',0,"Defaultoverview",)} Related Topics

AggModifiers constants

Constant	Data type	Description
CumulativeAgg	SmallInt	A cumulative summary that keeps a running total that extends from the start of the report to the end of the current group.
RegularAgg	SmallInt	A normal summary that considers all non-null values in the set, including duplicates.
UniqueAgg	SmallInt	A unique summary that counts only the unique non-null values in the set. Duplicates are ignored.

{button ,AL(`OPAL_CONST_CONSTYPES;',0,"Defaultoverview",)} **Related Topics**

BrowserOptions constants

Constant	Data type	Description
BrowseOptCreatePrompt	LongInt	Prompts the user for permission to create a file
BrowseOptFileMustExist	LongInt	Specifies that the user can type only names of existing files
BrowseOptNoNetButton	LongInt	Hides the network button on the dialog box
BrowseOptPathMustExist	LongInt	Specifies that the user can type only valid paths and filenames

{button ,AL(`OPAL_CONST_CONSTYPES;',0,"Defaultoverview",)} Related Topics

ButtonStyles constants

Constant	Data type	Description
BorlandButton	SmallInt	Gives a button the 3D look of buttons in Corel products
Windows3dButton	SmallInt	Gives a button the same look as 3D buttons in other Windows products
WindowsButton	SmallInt	Gives a button the same look as flat buttons in other Windows products

{button ,AL(`OPAL_CONST_CONSTYPES;',0,"Defaultoverview",)} Related Topics

ButtonTypes constants

Constant	Data type	Description
CheckboxType	SmallInt	Displays a button as a check box
PushButtonType	SmallInt	Displays a button as a push button
RadioButtonType	SmallInt	Displays a button as a radio button

{button ,AL(`OPAL_CONST_CONSTYPES;',0,"Defaultoverview",)} **Related Topics**

Color constants

Constant	Data type
Black	LongInt
Blue	LongInt
Brown	LongInt
DarkBlue	LongInt
DarkCyan	LongInt
DarkGray	LongInt
DarkGreen	LongInt
DarkMagenta	LongInt
DarkRed	LongInt
Gray	LongInt
Green	LongInt
LightBlue	LongInt
Magenta	LongInt
Red	LongInt
Translucent	LongInt
Transparent	LongInt
White	LongInt
Yellow	LongInt

{button ,AL(`OPAL_CONST_CONSTYPES;',0,"Defaultoverview",)} **Related Topics**

CompleteDisplay constants

Constant	Data type	Description
DisplayAll	SmallInt	Specifies CompleteDisplay for all field objects in the form
DisplayCurrent	SmallInt	Specifies CompleteDisplay for the current field object

{button ,AL(`OPAL_CONST_CONSTYPES;',0,"Defaultoverview",)} Related Topics

DataTransferCharset constants

Constant	Data type	Description
dtANSI	SmallInt	Specifies the ANSI character set
dtOEM	SmallInt	Specifies the OEM character set

{button ,AL(`OPAL_CONST_CONSTYPES;',0,"Defaultoverview",)} Related Topics

DataTransferDelimitCode constants

Constant	Data type	Description
dtDelimAllFields	SmallInt	Specifies to delimit all fields
dtDelimJustText	SmallInt	Specifies to delimit just text fields

{button ,AL(`OPAL_CONST_CONSTYPES;',0,"Defaultoverview",)} Related Topics

DataTransferFileType constants

Constant	Data type	Description
dt123V1	SmallInt	Specifies Lotus 123 (.WKS)
dt123V2	SmallInt	Specifies Lotus 123 (.WK1)
dtASCIIFixed	SmallInt	Specifies ASCII Fixed (BDE)
dtASCIIVar	SmallInt	Specifies ASCII Delimited
dtAuto	SmallInt	Automatically determine file type based on file extension
dtdBase3	SmallInt	Specifies Export to dBASE III + compatible
dtdBase4	SmallInt	Specifies Export to dBASE IV compatible
dtdBase5	SmallInt	Export to dBASE 5 compatible, Import any dBASE
dtdBaseAny	SmallInt	Import (or Export) any dBASE version
dtExcel4	SmallInt	Specifies Excel Version 3,4 (.XLS)
dtExcel5	SmallInt	Specifies Excel Version 5 (.XLS)
dtParadox3	SmallInt	Export to Corel Paradox 3 compatible
dtParadox4	SmallInt	Export to Corel Paradox 4 compatible
dtParadox5	SmallInt	Export to Corel Paradox 5 compatible
dtParadox7	SmallInt	Export to Corel Paradox 7 compatible
dtParadoxAny	SmallInt	Import (or Export) any Corel Paradox version
dtQPW1	SmallInt	Specifies Quattro Pro Windows 1, 5 (.WB1)
dtQPW6	SmallInt	Specifies Quattro Pro Windows 6 (.WB2)
dtQPW7	SmallInt	Specifies Quattro Pro Windows 95 (.WB3)
dtQuattro	SmallInt	Specifies Quattro DOS (.WKQ)
dtQuattroPro	SmallInt	Specifies Quattro Pro DOS (.WQ1)

{button ,AL(`OPAL_CONST_CONSTYPES;',0,"Defaultoverview",)} Related Topics

DateRangeTypes constants

Constant	Data type	Description
ByDay	SmallInt	Group report records by day
ByMonth	SmallInt	Group report records by month
ByQuarter	SmallInt	Group report records by quarter (3 months)
ByWeek	SmallInt	Group report records by week
ByYear	SmallInt	Group report records by year

{button ,AL(`OPAL_CONST_CONSTYPES;',0,"Defaultoverview",)} Related Topics

DesktopPreferenceTypes constants

Constant	Data type	Description
Section = prefProjectSection		
prefStartUpExpert	SmallInt	Run the Startup Expert each time Corel Paradox loads (Experts page)
prefTitleName	SmallInt	Title (General page)
prefExpertDefault	SmallInt	Run the experts when creating objects on documents (Experts page)
prefBackgroundName	SmallInt	Background bitmap (General page)
prefTileBitmap	SmallInt	Tile bitmap (General page)
prefSaveOnExit	SmallInt	Desktop state: Save on exit (General page)
prefRestoreDesktop	SmallInt	Desktop state: Restore on startup (General page)
prefSystemFont	SmallInt	Default system font (General page)
prefScreenPageDesk	SmallInt	On-screen size: Size to desktop (Forms/Reports page)
prefScreenPageWidth	SmallInt	On-screen size: Width (Forms/Reports page)
prefScreenPageHeight	SmallInt	On-screen size: Height (Forms/Reports page)
prefFormOpen	SmallInt	Open default: Open forms in design mode (Forms/Reports page)
prefReportOpen	SmallInt	Open default: Open reports in design mode (Forms/Reports page)
prefWarnOnDirChange	SmallInt	Don't show warning prompts when changing directories (Advanced page)
prefBitmapButtons	SmallInt	Changes to Corel-style buttons
prefAltKeyPadChars	SmallInt	Always use ALT + numeric keypad for character entry (Advanced page)
prefExpandBranchs	SmallInt	Indicate expandable directory branches (Advanced page)
prefScrollBarsInForms	SmallInt	Use scroll bars in form windows by default (Advanced page)
prefBlankAsZeroName	SmallInt	Treat blank fields as zeros (Database page)
prefRefreshRate	SmallInt	Refresh rate (seconds) (Database page)
prefExpertsOnCreate	SmallInt	Always use expert (New forms/reports)
prefUserLevel	SmallInt	ObjectPAL level (Developer Preferences: General page)
prefDeveloperMenu	SmallInt	Show developer menus (Developer Preferences: General page)
prefEnableControlBreak	SmallInt	Debugger settings: Enable CTRL + Break (Developer Preferences, General page)
Section = prefQbeSection		
prefAuxOpts	SmallInt	Generate auxiliary tables
prefSqlRunMode	SmallInt	Queries against remote tables (Query)
prefDefCheck	SmallInt	Default QBE check type
prefSqlconstrained	SmallInt	SQL answer constraints
Section = prefProjViewSection		
prefOpenOnStartup	SmallInt	Open Project Viewer on startup (Project Viewer settings: General page)

{button ,AL(`OPAL_CONST_CONSTYPES;',0,"Defaultoverview",)} **Related Topics**

ErrorReasons constants

Constant	Data type	Description
ErrorCritical	SmallInt	Displays a message in a modal dialog box
ErrorWarning	SmallInt	Displays a message in the status area

{button ,AL(`OPAL_CONST_CONSTYPES;',0,"Defaultoverview",)} Related Topics

EventErrorCodes constants

Constant	Data type	Description
Can_Arrive	SmallInt	Grants permission to arrive at an object
Can_Depart	SmallInt	Grants permission to leave an object
CanNotArrive	SmallInt	Refuses permission to arrive at an object (blocks the move)
CanNotDepart	SmallInt	Refuses permission to leave an object (blocks the move)

{button ,AL(`OPAL_CONST_CONSTYPES;',0,"Defaultoverview",)} **Related Topics**

ExecuteOptions constants

Constant	Data type	Description
ExeHidden	SmallInt	Hides the Application Window and passes activation to another window
ExeMinimized	SmallInt	Minimizes the Application Window and activates the top-level window in the window-manager's list
ExeShowMaximized	SmallInt	Activates the Application Window and displays it as a maximized window
ExeShowMinimized	SmallInt	Activates the Application Window and displays it minimized (as an icon)
ExeShowMinimizedNoActivate	SmallInt	Displays the application as an icon. The active window remains active.
ExeShowNoActivate	SmallInt	Displays the Application Window at its most recent size and position. The active window remains active.
ExeShowNormal	SmallInt	Activates and displays a window

{button ,AL(`OPAL_CONST_CONSTYPES;',0,"Defaultoverview",)} **Related Topics**

FieldDisplayTypes constants

Constant	Data type	Description
BitmapField	SmallInt	Enables a field object to display a bitmap
CheckboxField	SmallInt	Displays a field as a check box
ComboField	SmallInt	Displays a field as a drop-down edit list (also called a combo box)
EditField	SmallInt	Displays an unlabeled field
LabeledField	SmallInt	Displays a labeled field
ListField	SmallInt	Displays a list box
OleField	SmallInt	Enables a field to contain OLE data
RadioButtonField	SmallInt	Displays a field as one or more radio buttons

{button ,AL(`OPAL_CONST_CONSTYPES;',0,"Defaultoverview",)} **Related Topics**

FileBrowserFileTypes constants

Constant	Data type	Description
fbAllTables	LongInt	All table types supported by Corel Paradox (*.db, *.dbf, etc.)
fbASCII	LongInt	Delimited Text files (*.txt)
fbBitmap	LongInt	Bitmap graphics (*.bmp)
fbDBase	LongInt	dBASE tables (*.dbf)
fbDLL (8)	LongInt	Dynamic Link Libraries (*.dll)
fbDM (5.0)	LongInt	Data model files (*.dm)
fbExcel	LongInt	Excel worksheets (*.xls)
fbFiles	LongInt	All files (*.*)
fbForm	LongInt	Corel Paradox forms (*.fsl, *.fdl)
fbGraphic	LongInt	Graphic files (*.bmp, *.eps, *.gif, *.pcx, *.tif)
fbHTML (8)	LongInt	HTML files (*.htm)
fbHTMLTemplate (8)	LongInt	HTML template files (*.htt)
fbIni	LongInt	Initialization files (*.ini)
fbLibrary	LongInt	ObjectPAL libraries (*.lsl, *.ldl)
fbLotus1	LongInt	Lotus 1-2-3 version 1 worksheets (*.wks)
fbLotus2	LongInt	Lotus 1-2-3 version 2 worksheets (*.wk1)
fbOCX (8)	LongInt	ActiveX Controls (*.OCX)
fbParadox	LongInt	Corel Paradox tables (*.db)
fbQuattro	LongInt	Quattro worksheets (*.wkq)
fbQuattroPro	LongInt	Quattro Pro worksheets (*.wq1)
fbQuattroProWindows	LongInt	Quattro Pro for Windows notebooks (*.wb1)
fbQuery	LongInt	Query files (*.qbe)
fbReport	LongInt	Corel Paradox reports (*.rsl, *.rdl)
fbScreenStyle (5.0)	LongInt	Form style sheets (*.ft)
fbScript	LongInt	ObjectPAL scripts (*.ssl, *.sdl)
fbSimpleText (8)	LongInt	Text files (*.txt)
fbSQL (5.0)	LongInt	SQL files (*.sql)
fbTable	LongInt	All table types supported by Corel Paradox (*.db, *.dbf, etc.)
fbTableView	LongInt	Corel Paradox table view files (*.tv)
fbText	LongInt	All text files (*.txt, *.pxt, *.rtf)

{button ,AL(`OPAL_CONST_CONSTYPES;',0,"Defaultoverview",)} **Related Topics**

FontAttributes constants

Constant	Data type	Description
FontAttribBold	SmallInt	**bold**
FontAttribItalic	SmallInt	*italic*
FontAttribNormal	SmallInt	normal
FontAttribStrikeOut	SmallInt	~~strike out~~
FontAttribUnderline	SmallInt	underline

{button ,AL(`OPAL_CONST_CONSTYPES;',0,"Defaultoverview",)} Related Topics

FrameStyles constants

Constant	Data type	Description
DashDotDotFrame	SmallInt	A repeating sequence of one dash followed by two dots
DashDotFrame	SmallInt	A repeating sequence of one dash followed by one dot
DashedFrame	SmallInt	A repeating sequence of dashes
DottedFrame	SmallInt	A repeating sequence of dots
DoubleFrame	SmallInt	Two concentric boxes
Inside3DFrame	SmallInt	The frame appears pushed into the form
NoFrame	SmallInt	No frame
Outside3DFrame	SmallInt	The frame appears popped out of the form
ShadowFrame	SmallInt	A drop shadow
SolidFrame	SmallInt	A single solid box (no dashes or dots)
WideInsideDoubleFrame	SmallInt	Two concentric boxes; the inside box is wide
WideOutsideDoubleFrame	SmallInt	Two concentric boxes; the outside box is wide
Windows3dFrame (5.0)	SmallInt	Uses the default Windows 3D frame style
Windows3dGroup (5.0)	SmallInt	Uses the default Windows 3D group border

{button ,AL(`OPAL_CONST_CONSTYPES;',0,"Defaultoverview",)} **Related Topics**

General constants

Constant	Data type	Description
No	Logical	False
Off	Logical	False
On	Logical	True
Pi	Number	3.14159265358979323846
Yes	Logical	True

{button ,AL(`OPAL_CONST_CONSTYPES;',0,"Defaultoverview",)} Related Topics

GraphBindTypes constants

Constant	Data type	Description
Graph1DSummary	SmallInt	Specifies a one-dimensional summary chart and enables summary operators
Graph2DSummary	SmallInt	Specifies a two-dimensional summary chart and enables summary operators and group-by specification
GraphTabular	SmallInt	Specifies a tabular chart (default)

{button ,AL(`OPAL_CONST_CONSTYPES;',0,"Defaultoverview",)} Related Topics

GraphicMagnification constants

Constant	Data type	Description
Magnify100	SmallInt	Displays the chart at its actual size
Magnify200	SmallInt	Displays the chart at twice its actual size
Magnify25	SmallInt	Displays the chart at a quarter of its actual size
Magnify400	SmallInt	Displays the chart at four times its actual size
Magnify50	SmallInt	Displays the chart at half its actual size
MagnifyBestFit	SmallInt	Resizes the chart as necessary to fit the chart in the frame

{button ,AL(`OPAL_CONST_CONSTYPES;',0,"Defaultoverview",)} **Related Topics**

GraphLabelFormats constants

Constant	Data type	Description
GraphHideY	SmallInt	Hides the Y-value (2-D and 3-D Pie and Column charts only)
GraphPercent	SmallInt	Displays the Y-value as a percent (2-D and 3-D Pie and Column charts only)
GraphShowY	SmallInt	Displays the Y-value in the units used in the table (2-D and 3-D Pie and Column charts only)

{button ,AL(`OPAL_CONST_CONSTYPES;',0,"Defaultoverview",)} **Related Topics**

GraphLegendPosition constants

Constant	Data type	Description
LegendCenter	SmallInt	Displays the legend centered below the chart
LegendLeft	SmallInt	Displays the legend to the left of the chart

{button ,AL(`OPAL_CONST_CONSTYPES;',0,"Defaultoverview",)} **Related Topics**

GraphMarkers

Constant	Data type	Description
MarkerBoxedCross	SmallInt	Marker is a box with a cross in it
MarkerBoxed_Plus	SmallInt	Marker is a box with a plus sign in it
MarkerCross	SmallInt	Marker is a cross
MarkerFilledBox	SmallInt	Marker is a filled box
MarkerFilledCircle	SmallInt	Marker is a filled circle
MarkerFilledDownTriangle	SmallInt	Marker is a filled triangle pointing down
MarkerFilledTriangle	SmallInt	Marker is a filled triangle pointing up
MarkerFilledTriangles	SmallInt	Marker is two filled triangles pointing at each other
MarkerHollowBox	SmallInt	Marker is a hollow (unfilled) box
MarkerHollowCircle	SmallInt	Marker is a hollow circle
MarkerHollowDownTriangle	SmallInt	Marker is a hollow triangle pointing down
MarkerHollowTriangle	SmallInt	Marker is a hollow triangle pointing up
MarkerHollowTriangles	SmallInt	Marker is two hollow triangles pointing at each other
MarkerHorizontalLine	SmallInt	Marker is a horizontal line
MarkerPlus	SmallInt	Marker is a plus sign
MarkerVerticalLine	SmallInt	Marker is a vertical line

{button ,AL(`OPAL_CONST_CONSTYPES;',0,"Defaultoverview",)} Related Topics

GraphTypeOverRide

Constant	Data type	Description
GraphArea	SmallInt	Displays specified series as an area chart
GraphBar	SmallInt	Displays specified series as a bar chart
GraphDefault	SmallInt	Displays specified series in the default chart type
GraphLine	SmallInt	Displays specified series as a line chart

{button ,AL(`OPAL_CONST_CONSTYPES;',0,"Defaultoverview",)} **Related Topics**

GraphTypes

Constant	Data type	Description
Graph2DArea	SmallInt	2-dimensional area chart
Graph2DBar	SmallInt	2-dimensional bar chart
Graph2DColumns	SmallInt	2-dimensional column chart
Graph2DLine	SmallInt	2-dimensional line chart
Graph2DPie	SmallInt	2-dimensional pie chart
Graph2DRotatedBar	SmallInt	2-dimensional rotated bar chart
Graph2DStackedBar	SmallInt	2-dimensional stacked bar chart
Graph3DArea	SmallInt	3-dimensional area chart
Graph3DBar	SmallInt	3-dimensional bar chart
Graph3DColumns	SmallInt	3-dimensional column chart
Graph3DPie	SmallInt	3-dimensional pie chart
Graph3DRibbon	SmallInt	3-dimensional ribbon chart
Graph3DRotatedBar	SmallInt	3-dimensional rotated bar chart
Graph3DStackedBar	SmallInt	3-dimensional stacked bar chart
Graph3DStep	SmallInt	3-dimensional step chart
Graph3DSurface	SmallInt	3-dimensional surface chart
GraphXY	SmallInt	XY chart

{button ,AL(`OPAL_CONST_CONSTYPES;',0,"Defaultoverview",)} **Related Topics**

IdRanges

Constant	Data type	Description
UserAction	SmallInt	Minimum value for a user-defined action constant
UserActionMax	SmallInt	Maximum value for a user-defined action constant
UserError	SmallInt	Minimum value for a user-defined error constant
UserErrorMax	SmallInt	Maximum value for a user-defined error constant
UserMenu	SmallInt	Minimum value for a user-defined menu ID constant
UserMenuMax	SmallInt	Maximum value for a user-defined menu ID constant

{button ,AL(`OPAL_CONST_CONSTYPES;',0,"Defaultoverview",)} **Related Topics**

Keyboard constants

Constant	Data type	Description
VK_ADD	SmallInt	Add key
VK_APPS	SmallInt	Application property inspection key
VK_BACK	SmallInt	BACKSPACE key
VK_CANCEL	SmallInt	Used for control-break processing
VK_CAPITAL	SmallInt	Capital key
VK_CLEAR	SmallInt	Clear key
VK_CONTROL	SmallInt	CTRL key
VK_DECIMAL	SmallInt	Decimal key
VK_DELETE	SmallInt	DELETE key
VK_DIVIDE	SmallInt	Divide key
VK_DOWN	SmallInt	Down Arrow key
VK_END	SmallInt	END key
VK_ESCAPE	SmallInt	ESCAPE key
VK_EXECUTE	SmallInt	Execute key
VK_F1	SmallInt	F1 key
VK_F10	SmallInt	F10 key
VK_F11	SmallInt	F11 key
VK_F12	SmallInt	F12 key
VK_F13	SmallInt	F13 key
VK_F14	SmallInt	F14 key
VK_F15	SmallInt	F15 key
VK_F16	SmallInt	F16 key
VK_F2	SmallInt	F2 key
VK_F3	SmallInt	F3 key
VK_F4	SmallInt	F4 key
VK_F5	SmallInt	F5 key
VK_F6	SmallInt	F6 key
VK_F7	SmallInt	F7 key
VK_F8	SmallInt	F8 key
VK_F9	SmallInt	F9 key
VK_HELP	SmallInt	Help key
VK_HOME	SmallInt	HOME key
VK_INSERT	SmallInt	INSERT key
VK_LBUTTON	SmallInt	Left mouse button
VK_LEFT	SmallInt	Left Arrow key
VK_MBUTTON	SmallInt	Middle mouse button (3-button mouse)
VK_MENU	SmallInt	Menu key
VK_MULTIPLY	SmallInt	Multiply key
VK_NEXT	SmallInt	Page Down key
VK_NUMLOCK	SmallInt	NUM LOCK key
VK_NUMPAD0	SmallInt	Key pad 0 key
VK_NUMPAD1	SmallInt	Key pad 1 key
VK_NUMPAD2	SmallInt	Key pad 2 key
VK_NUMPAD3	SmallInt	Key pad 3 key
VK_NUMPAD4	SmallInt	Key pad 4 key
VK_NUMPAD5	SmallInt	Key pad 5 key

VK_NUMPAD6	SmallInt	Key pad 6 key
VK_NUMPAD7	SmallInt	Key pad 7 key
VK_NUMPAD8	SmallInt	Key pad 8 key
VK_NUMPAD9	SmallInt	Key pad 9 key
VK_PAUSE	SmallInt	Pause key
VK_PRINT	SmallInt	OEM specific
VK_PRIOR	SmallInt	Page Up key
VK_RBUTTON	SmallInt	Right mouse button
VK_RETURN	SmallInt	RETURN key
VK_RIGHT	SmallInt	Right Arrow key
VK_SELECT	SmallInt	Select key
VK_SEPARATOR	SmallInt	Separator key
VK_SHIFT	SmallInt	SHIFT key
VK_SNAPSHOT	SmallInt	Printscreen key for Windows 3.0 and later
VK_SPACE	SmallInt	SPACE
VK_SUBTRACT	SmallInt	Subtract key
VK_TAB	SmallInt	TAB key
VK_UP	SmallInt	Up Arrow key

{button ,AL(`OPAL_CONST_CONSTYPES;',0,"Defaultoverview",)} **Related Topics**

KeyBoardStates constants

Constant	Data type	Description
Alt	SmallInt	ALT is pressed
Control	SmallInt	CTRL is pressed
LeftButton	SmallInt	The left mouse button is clicked
RightButton	SmallInt	The right mouse button is clicked
Shift	SmallInt	SHIFT is pressed

{button ,AL(`OPAL_CONST_CONSTYPES;',0,"Defaultoverview",)} Related Topics

LibraryScope constants

Constant	Data type	Description
GlobalToDesktop	SmallInt	Makes variables in an ObjectPAL library available to one or more forms
PrivateToForm	SmallInt	Makes variables in an ObjectPAL library available to one form only

{button ,AL(`OPAL_CONST_CONSTYPES;',0,"Defaultoverview",)} **Related Topics**

LineEnd constants

Constant	Data type	Description
ArrowBothEnds	SmallInt	Adds arrows to both ends of a line (only if LineType = StraightLine)
ArrowOneEnd	SmallInt	Adds an arrow to the terminal end of a line (only if LineType = StraightLine)
NoArrowEnd	SmallInt	Displays a line without arrows

{button ,AL(`OPAL_CONST_CONSTYPES;',0,"Defaultoverview",)} Related Topics

LineStyles constants

Constant	Data type	Description
DashDotDotLine	SmallInt	A repeating sequence of one dash followed by two dots
DashDotLine	SmallInt	A repeating sequence of one dash followed by one dot
DashedLine	SmallInt	A repeating sequence of dashes
DottedLine	SmallInt	A repeating sequence of dots
NoLine	SmallInt	No line
SolidLine	SmallInt	An unbroken line

{button ,AL(`OPAL_CONST_CONSTYPES;',0,"Defaultoverview",)} **Related Topics**

LineThickness constants

Constant	Data type	Description
LWidth10Points	SmallInt	Specifies a thickness of 10 printer's points
LWidth1Point	SmallInt	Specifies a thickness of 1 printer's point
LWidth2Points	SmallInt	Specifies a thickness of 2 printer's points
LWidth3Points	SmallInt	Specifies a thickness of 3 printer's points
LWidth6Points	SmallInt	Specifies a thickness of 6 printer's points
LWidthHairline	SmallInt	Specifies a very thin line
LWidthHalfFoint	SmallInt	Specifies a thickness of one half of a printer's point

{button ,AL(`OPAL_CONST_CONSTYPES;',0,"Defaultoverview",)} **Related Topics**

LineTypes constants

Constant	Data type	Description
CurvedLine	SmallInt	Specifies a curved (elliptical) line
StraightLine	SmallInt	Specifies a straight line

{button ,AL(`OPAL_CONST_CONSTYPES;',0,"Defaultoverview",)} **Related Topics**

MailAddressTypes constants

Constant	Data type	Description
MailAddrTo	SmallInt	Specifies this address goes on the To line
MailAddrCC	SmallInt	Specifies this address goes on the CC line
MailAddrBC	SmallInt	Specifies this person gets a copy of the message, without letting anyone else see it (not be supported by all mail systems)

{button ,AL(`OPAL_CONST_CONSTYPES;',0,"Defaultoverview",)} Related Topics

MailReadOptions constants

Constant	Data type	Description
MailReadBodyAsFile	SmallInt	Write the message text to a temporary file and add it as the first attachment in the attachment list
MailReadEnvelopeOnly	SmallInt	Read the message header only. Do not copy file attachments to temporary files nor read message text. (Setting MailReadEnvelopeOnly enhances performance.)
MailReadPeek	SmallInt	Do not mark the message as read Marking a message as read affects its appearance in the user interface and generates a read receipt. If the mail system you are using does not support this, then MailReadPeek is ignored, and the message will be marked as read.
MailReadSuppressAttachments	SmallInt	Read mail message header and text, but do not copy file attachments. This option is ignored if using MailReadEnvelopeOnly. Specifying to MailReadSuppressAttachments enhances performance.

{button ,AL(`OPAL_CONST_CONSTYPES;',0,"Defaultoverview",)} Related Topics

MenuChoiceAttributes constants

Constant	Data type	Description
MenuChecked	SmallInt	Inserts a check mark before the menu item
MenuDisabled	SmallInt	Specifies that a menu item cannot be selected. Menu stays open
MenuEnabled	SmallInt	Specifies that a menu item can be selected. Menu closes
MenuGrayed	SmallInt	Displays a menu item in gray characters (dimmed)
MenuHilited	SmallInt	Highlights a menu item
MenuNotChecked	SmallInt	Displays a menu item without a check mark
MenuNotGrayed	SmallInt	Displays a menu item normally (not dimmed)
MenuNotHilited	SmallInt	Displays a menu item without a highlight

{button ,AL(`OPAL_CONST_CONSTYPES;',0,"Defaultoverview",)} **Related Topics**

MenuCommands constants

Constant	Data type	Description
MenuAddPage	SmallInt	Adds a page to a form
MenuAddWatch	SmallInt	Program, Add Watch
MenuAlignBar	SmallInt	Toggles the Align option from View, Toolbars
MenuAlignBottom	SmallInt	Format, Alignment, Align Bottom
MenuAlignCenter	SmallInt	Format, Alignment, Align Center
MenuAlignMiddle	SmallInt	Format, Alignment, Align Middle
MenuAlignLeft	SmallInt	Format, Alignment, Align Left
MenuAlignRight	SmallInt	Format, Alignment, Align Right
MenuAlignTop	SmallInt	Format, Alignment, Align Top
MenuBuild (4.5)	SmallInt	Reports when the desktop is building a form's menu
MenuCanClose	SmallInt	Asks for permission to continue after choosing Close (Control menu)
MenuChangedPriv (5.0)	SmallInt	Reports when the private directory has been changed. Forms that remain open after the change can use this information to make adjustments, as needed.
MenuChangedWork (5.0)	SmallInt	Reports when the working directory has been changed. Forms that remain open after the change can use this information to make adjustments, as needed.
MenuChangingPriv (5.0)	SmallInt	Reports when the private directory is about to change. Setting the error code to a nonzero value allows a form to stay open after the change; setting the error code to zero closes the form before changing the directory.
MenuChangingWork (5.0)	SmallInt	Reports when the working directory is about to change. Setting the error code to a nonzero value allows a form to stay open after the change; setting the error code to zero closes the form before changing the directory.
MenuCheckSyntax	SmallInt	Program, Check Syntax
MenuCompile	SmallInt	Program, Compile
MenuCompileWithDebug	SmallInt	Program, Compile With Debug
MenuControlClose	SmallInt	Close (Control menu)
MenuControlKeyMenu	SmallInt	Control menu was invoked by a pressing a key
MenuControlMaximize	SmallInt	Maximize (Control menu)
MenuControlMinimize	SmallInt	Minimize (Control menu)
MenuControlMouseMenu	SmallInt	Control menu was invoked by a mouse click
MenuControlMove	SmallInt	Move (Control menu)
MenuControlNextWindow	SmallInt	Next Window (Control menu)
MenuControlPrevWindow	SmallInt	Prev Window (Control menu)
MenuControlRestore	SmallInt	Restore (Control menu)
MenuControlSize	SmallInt	Size (Control menu)
MenuCopyToolbar	SmallInt	Edit, Copy to Toolbar
MenuDataModel	SmallInt	Format, Data Model
MenuDataModelDesigner	SmallInt	Tools, Data Model Designer
MenuDataModelNew	SmallInt	File, New, Data Model
MenuDeliver	SmallInt	Format, Deliver
MenuDesignBringFront	SmallInt	Format, Order, Bring to Front

MenuDesignDuplicate	SmallInt	Edit, Duplicate
MenuDesignGroup	SmallInt	Format, Group
MenuDesignLayout	SmallInt	Format, Layout
MenuDesignSendBack	SmallInt	Format, Order, Send to Back
MenuDmCommit	SmallInt	Accepts the changes you have made
MenuDmLoad	SmallInt	File, Open
MenuDmRestore	SmallInt	Cancels the change you have made
MenuDmSave	SmallInt	File, Save
MenuDmUnlink	SmallInt	Removes existing links
MenuEditCopy	SmallInt	Edit, Copy
MenuEditCopyTo	SmallInt	Edit, Copy To
MenuEditCut	SmallInt	Edit, Cut
MenuEditDelete	SmallInt	Edit, Delete
MenuEditLinks	SmallInt	For OLE objects only
MenuEditPaste	SmallInt	Edit, Paste
MenuEditUndo	SmallInt	Edit, Undo
MenuExpertsOpen	SmallInt	Tools, Experts
MenuFieldFilter	SmallInt	Right-click Filter on Field object
MenuFieldPicture	SmallInt	Right-click Picture... on unbound fields
MenuFileAliases	SmallInt	Tools, Alias Manager...
MenuFileAutoRefresh	SmallInt	Tools, Settings, Preferences, Database, Refresh Rate
MenuFileExit	SmallInt	File, Exit
MenuFileExport	SmallInt	File, Export
MenuFileImport	SmallInt	File, Import
MenuFileMultiBlankZero	SmallInt	Tools, Settings, Preferences, Treat blank fields as zeros
MenuFileMultiUserDrivers	SmallInt	Tools, Settings, Preferences, BDE, Database driver list
MenuFileMultiUserInfo	SmallInt	Tools, Settings, Preferences, Database
MenuFileMultiUserLock	SmallInt	Tools, Security, Set Locks
MenuFileMultiUserLockInfo	SmallInt	Tools, Security, Display Locks
MenuFileMultiUserRetry	SmallInt	Tools, Settings, Preferences, Database, Set Retry
MenuFileMultiUserUserName	SmallInt	Tools, Settings, Preferences, Database, User name
MenuFileMultiUserWho	SmallInt	Tools, Settings, Preferences, Database, Current user list
MenuFilePrint	SmallInt	File, Print
MenuFilePrinterSetup	SmallInt	File, Printer Setup
MenuFilePrivateDir	SmallInt	Tools, Settings, Preferences, Database, Private directory
MenuFileTableAdd	SmallInt	Tools, Utilities, Add
MenuFileTableCopy	SmallInt	Tools, Utilities, Copy
MenuFileTableDelete	SmallInt	Tools, Utilities, Delete
MenuFileTableEmpty	SmallInt	Tools, Utilities, Empty
MenuFileTableInfoStructure	SmallInt	Tools, Utilities, Info Structure
MenuFileTablePasswords	SmallInt	Tools, Security, Passwords
MenuFileTableRename	SmallInt	Tools, Utilities, Rename
MenuFileTableRestructure	SmallInt	Tools, Utilities, Restructure
MenuFileTableSort	SmallInt	Tools, Utilities, Sort
MenuFileTableSubtract	SmallInt	Tools, Utilities, Subtract

MenuFileWorkingDir	SmallInt	File, Working Directory
MenuFolderOpen	SmallInt	Tools, Project Viewer
MenuFormatBar	SmallInt	Toggles the Text Formatting option from View, Toolbars
MenuFormDesign	SmallInt	View, Design Form
MenuFormEditData	SmallInt	View, Edit Data
MenuFormFieldView	SmallInt	View, Field View
MenuFormFilter	SmallInt	Format, Filter
MenuFormMemoView	SmallInt	View, Memo View
MenuFormNew	SmallInt	Opens a new form
MenuFormOpen	SmallInt	Opens a form
MenuFormOrderRange	SmallInt	Format, Filter
MenuFormPageFirst	SmallInt	View, Page, First
MenuFormPageGoto	SmallInt	View, Page, Go To
MenuFormPageLast	SmallInt	View, Page, Last
MenuFormPageNext	SmallInt	View, Page, Next
MenuFormPagePrevious	SmallInt	View, Page, Previous
MenuFormPersistView	SmallInt	View, Persistent Field View
MenuFormShowDeleted	SmallInt	View, Show Deleted
MenuFormTableView	SmallInt	View, Table View
MenuFormView	SmallInt	View, View Data
MenuFormViewData (5.0)	SmallInt	View, View Data
MenuHelpAbout	SmallInt	An Introduction to the available help
MenuHelpCoach (5.0)	SmallInt	Displays an expert coach to help
MenuHelpContents	SmallInt	Displays the help table of contents
MenuHelpKeyboard	SmallInt	Displays keyboard help
MenuHelpSearch	SmallInt	Displays the ObjectPAL references
MenuHelpSupport	SmallInt	Displays Technical Support info
MenuHelpToolbar (5.0)	SmallInt	Displays the Toolbar help
MenuHelpUsingHelp	SmallInt	Displays an introduction to the help available
MenuInit	SmallInt	Generated by clicking a menu items
MenuInsertObject	SmallInt	Edit, Insert Object (For OLE objects only)
MenuInsOleControl	SmallInt	Tools, Register
MenuLibraryNew	SmallInt	Opens a new library
MenuLibraryOpen	SmallInt	Opens a library
MenuNextWarning	SmallInt	Search, Next Warning
MenuNoteBookAddPage	SmallInt	Insert, Page
MenuNoteBookFirstPage	SmallInt	View, Page, First
MenuNoteBookLastPage	SmallInt	View, Page, Last
MenuNoteBookNextPage	SmallInt	View, Page, Next
MenuNoteBookPriorPage	SmallInt	View, Page, Previous
MenuNoteBookRotate	SmallInt	Format, Rotate Pages
MenuObjectBar	SmallInt	Toggles the Object option from View, Toolbars
MenuOpenProjectView (5.0)	SmallInt	Tools, Project Viewer
MenuPageLayout	SmallInt	Format, Layout
MenuPALSave	SmallInt	Edit, Save Debug State
MenuPasteFrom	SmallInt	Edit, Paste From
MenuPasteLink	SmallInt	Edit, Paste Link
MenuProjAdd	SmallInt	Add Reference button (Standard toolbar)
MenuProjDelete	SmallInt	Remove Reference button (Standard toolbar)

MenuPropertiesBandLabels	SmallInt	View, Band Labels
MenuPropertiesCurrent	SmallInt	Edit, Current Object
MenuPropertiesCurrentDialog	SmallInt	Edit, Current Object
MenuPropertiesDesigner	SmallInt	View, Design Form
MenuPropertiesDesktop	SmallInt	View, View Data
MenuPropertiesExpandedRuler	SmallInt	Expanded Ruler (Designer Preferences options)
MenuPropertiesFormRestoreDefaults	SmallInt	Restores the default settings for a form
MenuPropertiesFormSaveDefaults	SmallInt	Saves the default settings for a form
MenuPropertiesGroupRepeats	SmallInt	Format, Properties, Remove group repeat
MenuPropertiesHorizontalRuler	SmallInt	Horizontal Ruler (Designer Preferences options)
MenuPropertiesMethods	SmallInt	Object Explorer
MenuPropertiesShowGrid	SmallInt	View, Grid
MenuPropertiesSizeandPos	SmallInt	View, Size and Position
MenuPropertiesSizeToFit	SmallInt	Format, Properties
MenuPropertiesSnapToGrid	SmallInt	Format, Snap to Grid
MenuPropertiesStyleSheet	SmallInt	Format, Style Sheet
MenuPropertiesVerticalRuler	SmallInt	Vertical Ruler (Designer Preferences options)
MenuPropertiesWindow	SmallInt	Displays the Window Style
MenuPropertiesZoom100	SmallInt	View, Zoom, 100%
MenuPropertiesZoom200	SmallInt	View, Zoom, 200%
MenuPropertiesZoom25	SmallInt	View, Zoom, 25%
MenuPropertiesZoom400	SmallInt	View, Zoom, 400%
MenuPropertiesZoom50	SmallInt	View, Zoom, 50%
MenuPropertiesZoomBestFit	SmallInt	View, Zoom, Best Fit
MenuPropertiesZoomFitHeight	SmallInt	View, Zoom, Fit Height
MenuPropertiesZoomFitWidth	SmallInt	View, Zoom, Fit Width
MenuQueryNew	SmallInt	Opens a new query
MenuQueryOpen	SmallInt	Opens an existing query
MenuQBEDoJoin	SmallInt	Join Tables button (Query design window toolbar)
MenuQBEProperties	SmallInt	Query, Properties
MenuQBEShowSQL	SmallInt	View, Show SQL
MenuQBESortAnswer	SmallInt	Query, Properties: Sort
MenuQuickForm	SmallInt	Tools, Quick Form
MenuQuickGraph	SmallInt	Tools, Quick Chart
MenuQuickReport	SmallInt	Tools, Quick Report
MenuQuickXTab	SmallInt	Tools, Quick Crosstab
MenuRecordCancel	SmallInt	Record, Cancel Changes
MenuRecordDelete	SmallInt	Record, Delete
MenuRecordFastBackward	SmallInt	Record, Previous Set
MenuRecordFastForward	SmallInt	Record, Next Set
MenuRecordFirst	SmallInt	Record, First
MenuRecordInsert	SmallInt	Record, Insert
MenuRecordLast	SmallInt	Record, Last
MenuRecordLocateNext	SmallInt	Record, Locate Next
MenuRecordLocateRecordNumber	SmallInt	Record, Locate, Record Number
MenuRecordLocateSearchAndReplace	SmallInt	Record, Locate, and Replace
MenuRecordLocateValue	SmallInt	Record, Locate, Value
MenuRecordLock	SmallInt	Record, Lock
MenuRecordLookup	SmallInt	Record, Lookup Help

MenuRecordMove	SmallInt	Record, Move Help
MenuRecordNext	SmallInt	Record, Next
MenuRecordPost	SmallInt	Record, Post/Keep Locked
MenuRecordPrevious	SmallInt	Record, Previous
MenuReportAddBand	SmallInt	Insert, Group Band
MenuReportNew	SmallInt	Opens a new report
MenuReportOpen	SmallInt	Opens a report
MenuReportPageFirst	SmallInt	View, Page, First
MenuReportPageGoto	SmallInt	View, Page, Go To
MenuReportPageLast	SmallInt	View, Page, Last
MenuReportPageNext	SmallInt	View, Page, Next
MenuReportPagePrevious	SmallInt	View, Page, Previous
MenuReportPrintDesign	SmallInt	Prints the file
MenuReportRestartOpts	SmallInt	Format, Restart Options
MenuRotatePage	SmallInt	Format, Rotate Pages
MenuSave	SmallInt	File, Save
MenuSaveAs	SmallInt	File, Save As...
MenuSaveCrossTab	SmallInt	Edit, Save Crosstab (must have a defined crosstab on a runtime form)
MenuScriptNew	SmallInt	Opens a new script
MenuScriptOpen	SmallInt	Opens a script
MenuSearchText	SmallInt	Edit, Search Text
MenuSelectAll	SmallInt	Edit, Select All
MenuSetBreakPoint	SmallInt	Program, Toggle Breakpoint
MenuSizeMaxHeight	SmallInt	Format, Size, Maximum Height
MenuSizeMaxWidth	SmallInt	Format, Size, Maximum Width
MenuSizeMinHeight	SmallInt	Format, Size, Minimum Height
MenuSizeMinWidth	SmallInt	Format, Size, Minimum Width
MenuSpaceHorz	SmallInt	Format, Spacing, Horizontal
MenuSpaceVert	SmallInt	Format, Spacing, Vertical
MenuSQLFileNew	SmallInt	Opens a new SQL File
MenuSQLFileOpen	SmallInt	Opens an SQL File
MenuStackPages	SmallInt	Stacks the pages in a form
MenuStepInto	SmallInt	Program, Step Into
MenuStepOver	SmallInt	Program, Step Over
MenuTableNew	SmallInt	Opens a new table
MenuTableOpen	SmallInt	Opens a table
MenuTileHorizontal	SmallInt	Tiles a form's pages side-by-side
MenuTileVertical	SmallInt	Tiles a form's pages, top and bottom
MenuViewBreakPoints	SmallInt	View, Breakpoints
MenuViewDebugger	SmallInt	View, Debugger
MenuViewMethods	SmallInt	View, ObjectPAL Quick Lookup: Types and Methods
MenuViewSource	SmallInt	View, Source
MenuViewStack	SmallInt	View, Call Stack
MenuViewTracer	SmallInt	View, Tracer
MenuViewTypes	SmallInt	View, ObjectPAL Quick Lookup: Types and Methods
MenuViewWatch	SmallInt	View, Watch
MenuWindowArrangeIcons	SmallInt	Window, Arrange Icons

MenuWindowCascade	SmallInt	Window, Cascade
MenuWindowCloseAll	SmallInt	Window, Close All
MenuWindowTile	SmallInt	Window, Tile

{button ,AL(`OPAL_CONST_CONSTYPES;',0,"Defaultoverview",)} **Related Topics**

MenuReasons constants

Constant	Data type	Description
MenuControl	SmallInt	Triggered by choosing an item from the control menu
MenuDesktop	SmallInt	Triggered by choosing an item from a built-in Corel Paradox menu
MenuNormal	SmallInt	Triggered by choosing an item from a custom ObjectPAL menu or by clicking a Toolbar button

{button ,AL(`OPAL_CONST_CONSTYPES;',0,"Defaultoverview",)} Related Topics

MouseShapes constants

Constant	Data type	Description
MouseArrow	LongInt	Standard pointer arrow
MouseCross	LongInt	Pointer is a cross
MouseIBeam	LongInt	Pointer is an I-beam (text insertion cursor)
MouseSize	LongInt	Pointer is normal size
MouseSizeNWSE	LongInt	Pointer is two-headed arrow pointing Northwest-Southeast
MouseSizeNESW	LongInt	Pointer is two-headed arrow pointing Northeast-Southwest
MouseSizeWE	LongInt	Pointer is two-headed arrow pointing East-West
MouseSizeNS	LongInt	Pointer is two-headed arrow pointing North-South
MouseNo	LongInt	Pointer is the international symbol for NO
MouseHand	LongInt	Pointer is a hand
MouseHelp	LongInt	Pointer is the standard arrow and a question mark
MouseDrag	LongInt	Pointer is the standard document drag and drop
MouseUpArrow	LongInt	Pointer is an arrow pointing up
MouseWait	LongInt	Pointer is an hourglass

{button ,AL(`OPAL_CONST_CONSTYPES;',0,"Defaultoverview",)} **Related Topics**

MoveReasons constants

Constant	Data type	Description
PalMove	SmallInt	Caused by an ObjectPAL statement
RefreshMove	SmallInt	Caused when data is updated, for example, by scrolling through a table
ShutDownMove	SmallInt	Caused when the form closes
StartupMove	SmallInt	Caused when the form opens
UserMove	SmallInt	Caused by the user

{button ,AL(`OPAL_CONST_CONSTYPES;',0,"Defaultoverview",)} Related Topics

PageTilingOptions constants

Constant	Data type	Description
StackPages	SmallInt	Pages are stacked one on top of the other
TileHorizontal	SmallInt	Pages are tiled horizontally
TileVertical	SmallInt	Pages are tiled vertically

{button ,AL(`OPAL_CONST_CONSTYPES;',0,"Defaultoverview",)} **Related Topics**

PatternStyles

Constant	Data type
BricksPattern	SmallInt
CrosshatchPattern	SmallInt
DiagonalCrosshatchPattern	SmallInt
DottedLinePattern	SmallInt
EmptyPattern	SmallInt
FuzzyStripesDownPattern	SmallInt
HeavyDotPattern	SmallInt
HorizontalLinesPattern	SmallInt
LatticePattern	SmallInt
LeftDiagonalLinesPattern	SmallInt
LightDotPattern	SmallInt
MaximumDotPattern	SmallInt
MediumDotPattern	SmallInt
RightDiagonalLinesPattern	SmallInt
ScalesPattern	SmallInt
StaggeredDashesPattern	SmallInt
ThickHorizontalLinesPattern	SmallInt
ThickStripesDownPattern	SmallInt
ThickStripesUpPattern	SmallInt
ThickVerticalLinesPattern	SmallInt
VerticalLinesPattern	SmallInt
VeryHeavyDotPattern	SmallInt
WeavePattern	SmallInt
ZigZagPattern	SmallInt

{button ,AL(`OPAL_CONST_CONSTYPES;',0,"Defaultoverview",)} **Related Topics**

PrintColor constants

Constant	Data type	Description
prnColor	LongInt	Print in color (color printers only)
prnMonochrome	LongInt	Print in monochrome

{button ,AL(`OPAL_CONST_CONSTYPES;',0,"Defaultoverview",)} **Related Topics**

PrintDuplex constants

Constant	Data type	Description
prnHorizontal	LongInt	Double-sided printing where the left and right edges of consecutive pages can be bound (also called bind on edge printing)
prnSimplex	LongInt	Single-sided printing
prnVertical	LongInt	Double-sided printing where the top and bottom edges of consecutive pages can be bound (also called bind on top printing)

{button ,AL(`OPAL_CONST_CONSTYPES;',0,"Defaultoverview",)} Related Topics

PrinterOrientation constants

Constant	Data type	Description
prnLandscape	LongInt	Landscape (long) orientation
prnPortrait	LongInt	Portrait (tall) orientation

{button ,AL(`OPAL_CONST_CONSTYPES;',0,"Defaultoverview",)} **Related Topics**

PrinterSizes constants

Constant	Data type	Description
prn10x14	LongInt	10 by 14 inches
prn11x17	LongInt	11 by 17 inches
prnA3	LongInt	A3 297 x 420 mm
prnA4	LongInt	A4 210 x 297 mm
prnA4Small	LongInt	A4 Small 210 x 297 mm
prnA5	LongInt	A5 148 x 210 mm
prnB4	LongInt	B4 250 x 354
prnB5	LongInt	B5 182 x 257 mm
prnCSheet	LongInt	C size sheet
prnDSheet	LongInt	D size sheet
prnEnv9	LongInt	Envelope #9 3 7/8 x 8 7/8 inches
prnEnv10	LongInt	Envelope #10 4 1/8 x 9 1/2 inches
prnEnv11	LongInt	Envelope #11 4 1/2 x 10 3/8 inches
prnEnv12	LongInt	Envelope #12 4 3/4 x 11 inches
prnEnv14	LongInt	Envelope #14 5 x 11 1/2 inches
prnEnvB4	LongInt	Envelope B4 250 x 353 mm
prnEnvB5	LongInt	Envelope B5 176 x 250 mm
prnEnvB6	LongInt	Envelope B6 176 x 125 mm
prnEnvC3	LongInt	Envelope C3 324 x 458 mm
prnEnvC4	LongInt	Envelope C4 229 x 324 mm
prnEnvC5	LongInt	Envelope C5 162 x 229 mm
prnEnvC6	LongInt	Envelope C6 114 x 162 mm
prnEnvC65	LongInt	Envelope C65 114 x 229 mm
prnEnvDL	LongInt	Envelope DL 110 x 220mm
prnEnvItaly	LongInt	Envelope 110 x 230 mm
prnEnvMonarch	LongInt	Envelope Monarch 3.875 x 7.5 inches
prnEnvPersonal	LongInt	6 3/4 Envelope 3 5/8 x 6 1/2 inches
prnESheet	LongInt	E size sheet
prnExecutive	LongInt	Executive 7 1/4 x 10 1/2 inches
prnFanfoldLegalGerman	LongInt	German Legal Fanfold 8 1/2 x 13 inches
prnFanfoldStandardGerman	LongInt	German Std Fanfold 8 1/2 x 12 inches
prnFanfoldUS	LongInt	US Std Fanfold 14 7/8 x 11 inches
prnFolio	LongInt	Folio 8 1/2 x 13 inches
prnLedger	LongInt	Ledger 17 x 11 inches
prnLegal	LongInt	Legal 8 1/2 x 14 inches
prnLetter	LongInt	Letter 8 1/2 x 11 inches
prnLetterSmall	LongInt	Letter Small 8 1/2 x 11 inches
prnNote	LongInt	Note 8 1/2 x 11 inches
prnQuarto	LongInt	Quarto 215 x 275 mm
prnStatement	LongInt	Statement 5 1/2 x 8 1/2 inches
prnTabloid	LongInt	Tabloid 11 x 17 inches

{button ,AL(`OPAL_CONST_CONSTYPES;',0,"Defaultoverview",)} **Related Topics**

PrintQuality constants

Constant	Data type	Description
prnDraft	LongInt	Draft quality (lowest quality, fastest print time)
prnHigh	LongInt	High quality (highest quality, slowest print time)
prnLow	LongInt	Low quality
prnMedium	LongInt	Medium quality

{button ,AL(`OPAL_CONST_CONSTYPES;',0,"Defaultoverview",)} **Related Topics**

PrintSources constants

Constant	Data type	Description
prnAuto	LongInt	Paper source selected automatically
prnCassette	LongInt	Cassette
prnEnvelope	LongInt	Envelope, automatic feed
prnEnvManual	LongInt	Envelope, manual feed
prnLargeCapacity	LongInt	Large capacity paper source
prnLargeFmt	LongInt	Large format paper source
prnLower	LongInt	Lower paper tray
prnManual	LongInt	Manual feed
prnMiddle	LongInt	Middle paper tray
prnOnlyOne	LongInt	Single paper tray
prnSmallFmt	LongInt	Small format paper source
prnTractor	LongInt	Tractor feed paper
prnUpper	LongInt	Upper paper tray

{button ,AL(`OPAL_CONST_CONSTYPES;',0,"Defaultoverview",)} **Related Topics**

QueryRestartOptions constants

Constant	Data type	Description
QueryDefault	SmallInt	Use the options specified interactively using the Query Restart Options dialog box
QueryLock	SmallInt	Lock all other users out of the tables needed while the query is running. If Corel Paradox cannot lock a table, it does not run the query. This is the least polite to other users. You must wait until all the locks can be secured before the query will run.
QueryNoLock	SmallInt	Run the query even if someone changes the data while it's running.
QueryRestart	SmallInt	Start the query over. Specify QueryRestart when you want to make sure you get a snapshot of the data as it existed at some instant. Another user might change the data after the query is completed but before the Answer table is displayed, but at least you got a snapshot. This is just the nature of multi-user work.

{button ,AL(`OPAL_CONST_CONSTYPES;',0,"Defaultoverview",)} Related Topics

RasterOperations constants

Constant	Data type	Description
MergePaint	LongInt	Inverts the source graphic and combines it with the destination using the Boolean OR operator
NotSourceCopy	LongInt	Inverts the source graphic and copies it to the destination
NotSourceErase	LongInt	Combines the source graphic and the destination and inverts the result using the Boolean OR operator
SourceAnd	LongInt	Combines the source graphic and the destination using the Boolean AND operator
SourceCopy	LongInt	Copies an unchanged source graphic to the destination
SourceErase	LongInt	Inverts the destination and combines it with the source graphic using the Boolean AND operator
SourceInvert	LongInt	Combines the source graphic and the destination using the Boolean XOR operator
SourcePaint	LongInt	Combines the source graphic and the destination using the Boolean OR operator

{button ,AL(`OPAL_CONST_CONSTYPES;',0,"Defaultoverview",)} **Related Topics**

RegistryKeyType constants

Constant	Data type	Description
regKeyClassesRoot	LongInt	Alias to HKEY_CLASSES_ROOT in the Registry
regKeyCurrentUser	LongInt	Alias to HKEY_CURRENT_USER in the Registry
regKeyLocalMachine	LongInt	Alias to HKEY_LOCAL_MACHINE in the Registry
regKeyUser	LongInt	Alias to HKEY_USERS in the Registry

{button ,AL(`OPAL_CONST_CONSTYPES;',0,"Defaultoverview",)} Related Topics

ReportOrientation constants

Constant	Data type	Description
PrintDefault	SmallInt	Use the current Windows default orientation
PrintLandscape	SmallInt	Use landscape (long) orientation
PrintPortrait	SmallInt	Use portrait (tall) orientation

{button ,AL(`OPAL_CONST_CONSTYPES;',0,"Defaultoverview",)} **Related Topics**

ReportPrintPanel constants

Constant	Data type	Description
PrintClipToWidth	SmallInt	Clips (trims) all data that does not fit across the page (within the margins)
PrintHorizontalPanel	SmallInt	Prints additional pages as needed to fit all the data. Each of these pages immediately follows the page it extends.
PrintOverflowPages	SmallInt	Same as PrintHorizontalPanel
PrintVerticalPanel	SmallInt	Creates a secondary page for each page of the report, even if it doesn't overflow

{button ,AL(`OPAL_CONST_CONSTYPES;',0,"Defaultoverview",)} Related Topics

ReportPrintRestart constants

Constant	Data type	Description
PrintFromCopy	SmallInt	Prints the report from copies of the tables in the report's data model
PrintLock	SmallInt	Locks tables in the report's data model before printing
PrintNoLock	SmallInt	Prints without locking tables in the report's table model
PrintRestart	SmallInt	Restarts print job when data changes in tables in the report's data model
PrintReturn	SmallInt	Cancel the print job when data changes in tables in the report's data model

{button ,AL(`OPAL_CONST_CONSTYPES;',0,"Defaultoverview",)} Related Topics

RestructureOperations constants

Constant	Data type	Description
crModify	SmallInt	Modify an existing field
crAdd	SmallInt	Add a new field
crDrop	SmallInt	Drop (delete) an existing field

{button ,AL(`OPAL_CONST_CONSTYPES;',0,"Defaultoverview",)} **Related Topics**

SpecialFieldTypes constants

Constant	Data type	Description
DateField	SmallInt	Displays the current system date
NofFieldsField	SmallInt	Displays the number of fields in the current table
NofPagesField	SmallInt	Displays the number of pages in the current form or report
NofRecsField	SmallInt	Displays the number of records in the current table
PageNumField	SmallInt	Displays the current page number
RecordNoField	SmallInt	Displays the active record number
TableNameField	SmallInt	Displays the name of the current table
TimeField	SmallInt	Displays the current system time

{button ,AL(`OPAL_CONST_CONSTYPES;',0,"Defaultoverview",)} **Related Topics**

StatusReasons constants

Constant	Data type	Description
ModeWindow1	SmallInt	The Status Bar area second from the left
ModeWindow2	SmallInt	The Status Bar area third from the left
ModeWindow3	SmallInt	The rightmost Status Bar area
StatusWindow	SmallInt	The leftmost (and largest) Status Bar area

{button ,AL(`OPAL_CONST_CONSTYPES;',0,"Defaultoverview",)} **Related Topics**

TableFrameStyles constants

Constant	Data type	Description
tf3D	SmallInt	Table frame has a 3D frame
tfDoubleLine	SmallInt	Table frame has a double-box frame
tfNoGrid	SmallInt	Table frame has no grid
tfSingleLine	SmallInt	Table frame has a box frame
tfTripleLine	SmallInt	Table frame has a triple-box frame

{button ,AL(`OPAL_CONST_CONSTYPES;',0,"Defaultoverview",)} **Related Topics**

TextAlignment constants

Constant	Data type	Description
TextAlignBottom	SmallInt	Bottom of text is aligned (table window only)
TextAlignCenter	SmallInt	Text is centered horizontally
TextAlignJustify	SmallInt	Text is justified right and left (does not apply to table window)
TextAlignLeft	SmallInt	Text is left-justified
TextAlignRight	SmallInt	Text is right-justified
TextAlignTop	SmallInt	Top of text is aligned (table window only)
TextAlignVCenter	SmallInt	Text is centered vertically (table window only)

{button ,AL(`OPAL_CONST_CONSTYPES;',0,"Defaultoverview",)} Related Topics

TextDesignSizing constants

Constant	Data type	Description
TextFixedSized	SmallInt	Text box does not change size
TextGrowOnly	SmallInt	Text box grows to accommodate text
TextSizeToFit	SmallInt	Text box grows or shrinks as necessary to accommodate text

{button ,AL(`OPAL_CONST_CONSTYPES;',0,"Defaultoverview",)} **Related Topics**

TextSpacing constants

Constant	Data type	Description
TextDoubleSpacing	SmallInt	2 lines
TextDoubleSpacing2	SmallInt	2.5 lines
TextSingleSpacing	SmallInt	1 line
TextSingleSpacing2	SmallInt	1.5 lines
TextTripleSpacing	SmallInt	3 lines

{button ,AL(`OPAL_CONST_CONSTYPES;',0,"Defaultoverview",)} Related Topics

ToolbarBitmap constants

Constant	Data type	Description
BitmapAddBand	SmallInt	System bitmap
BitmapAddTable	SmallInt	System bitmap
BitmapAddToCat	SmallInt	System bitmap
BitmapAlignBottom	SmallInt	System bitmap
BitmapAlignCenter	SmallInt	System bitmap
BitmapAlignLeft	SmallInt	System bitmap
BitmapAlignMiddle	SmallInt	System bitmap
BitmapAlignRight	SmallInt	System bitmap
BitmapAlignTop	SmallInt	System bitmap
BitmapBookTool	SmallInt	System bitmap
BitmapBoxTool	SmallInt	System bitmap
BitmapBringToFront	SmallInt	System bitmap
BitmapButtonTool	SmallInt	System bitmap
BitmapCancel	SmallInt	System bitmap
BitmapChartTool	SmallInt	System bitmap
BitmapChkSyntax	SmallInt	System bitmap
BitmapCoEdit	SmallInt	System bitmap
BitmapCompile	SmallInt	System bitmap
BitmapDataBegin	SmallInt	System bitmap
BitmapDataEnd	SmallInt	System bitmap
BitmapDataModel	SmallInt	System bitmap
BitmapDataNextRecord	SmallInt	System bitmap
BitmapDataNextSet	SmallInt	System bitmap
BitmapDataPriorRecord	SmallInt	System bitmap
BitmapDataPriorSet	SmallInt	System bitmap
BitmapDelTable	SmallInt	System bitmap
BitmapDesignMode	SmallInt	System bitmap
BitmapDoJoin	SmallInt	System bitmap
BitmapDuplicate	SmallInt	System bitmap
BitmapEditAnswer	SmallInt	System bitmap
BitmapEditCopy	SmallInt	System bitmap
BitmapEditCut	SmallInt	System bitmap
BitmapEditPaste	SmallInt	System bitmap
BitmapEllipseTool	SmallInt	System bitmap
BitmapFieldTool	SmallInt	System bitmap
BitmapFilter	SmallInt	System bitmap
BitmapFirstPage	SmallInt	System bitmap
BitmapFldView	SmallInt	System bitmap
BitmapFontAttribBold	SmallInt	System bitmap
BitmapFontAttribItalic	SmallInt	System bitmap
BitmapFontAttribStrikeout	SmallInt	System bitmap
BitmapFontAttribUnderline	SmallInt	System bitmap
BitmapGotoPage	SmallInt	System bitmap
BitmapGraphicTool	SmallInt	System bitmap
BitmapGroup	SmallInt	System bitmap
BitmapHelp	SmallInt	System bitmap

BitmapHSpacing	SmallInt	System bitmap
BitmapLastPage	SmallInt	System bitmap
BitmapLineSpace1	SmallInt	System bitmap
BitmapLineSpace15	SmallInt	System bitmap
BitmapLineSpace2	SmallInt	System bitmap
BitmapLineSpace25	SmallInt	System bitmap
BitmapLineSpace3	SmallInt	System bitmap
BitmapLineSpace35	SmallInt	System bitmap
BitmapLineTool	SmallInt	System bitmap
BitmapLinkDm	SmallInt	System bitmap
BitmapLoadDm	SmallInt	System bitmap
BitmapMaxHeight	SmallInt	System bitmap
BitmapMaxWidth	SmallInt	System bitmap
BitmapMinHeight	SmallInt	System bitmap
BitmapMinWidth	SmallInt	System bitmap
BitmapNextPage	SmallInt	System bitmap
BitmapNextWarn	SmallInt	System bitmap
BitmapObjectTree	SmallInt	System bitmap
BitmapOk	SmallInt	System bitmap
BitmapOleTool	SmallInt	System bitmap
BitmapOpenExpert	SmallInt	System bitmap
BitmapOpenForm	SmallInt	System bitmap
BitmapOpenLibrary	SmallInt	System bitmap
BitmapOpenProject	SmallInt	System bitmap
BitmapOpenQbe	SmallInt	System bitmap
BitmapOpenReport	SmallInt	System bitmap
BitmapOpenScript	SmallInt	System bitmap
BitmapOpenSql	SmallInt	System bitmap
BitmapOpenTable	SmallInt	System bitmap
BitmapOpenTutor	SmallInt	System bitmap
BitmapPageBreak	SmallInt	System bitmap
BitmapPickTool	SmallInt	System bitmap
BitmapPrevPage	SmallInt	System bitmap
BitmapPrint	SmallInt	System bitmap
BitmapQuickForm	SmallInt	System bitmap
BitmapQuickGraph	SmallInt	System bitmap
BitmapQuickReport	SmallInt	System bitmap
BitmapQuickXTab	SmallInt	System bitmap
BitmapRecordTool	SmallInt	System bitmap
BitmapRemoveFromCat	SmallInt	System bitmap
BitmapRestructure	SmallInt	System bitmap
BitmapRun	SmallInt	System bitmap
BitmapSave	SmallInt	System bitmap
BitmapSaveDm	SmallInt	System bitmap
BitmapSendToBack	SmallInt	System bitmap
BitmapSetBreak	SmallInt	System bitmap
BitmapSetOrgin	SmallInt	System bitmap
BitmapSetWatch	SmallInt	System bitmap
BitmapShowSQL	SmallInt	System bitmap

BitmapSortAnswer	SmallInt	System bitmap
BitmapSpeedExit	SmallInt	System bitmap
BitmapSrchNext	SmallInt	System bitmap
BitmapSrchValue	SmallInt	System bitmap
BitmapStepInto	SmallInt	System bitmap
BitmapStepOver	SmallInt	System bitmap
BitmapStop	SmallInt	System bitmap
BitmapTableFrameTool	SmallInt	System bitmap
BitmapTButton	SmallInt	System bitmap
BitmapTComboBox	SmallInt	System bitmap
BitmapTextCenter	SmallInt	System bitmap
BitmapTextJustify	SmallInt	System bitmap
BitmapTextLeft	SmallInt	System bitmap
BitmapTextRight	SmallInt	System bitmap
BitmapTextTool	SmallInt	System bitmap
BitmapTGuage	SmallInt	System bitmap
BitmapTHeader	SmallInt	System bitmap
BitmapTListBox	SmallInt	System bitmap
BitmapTSpinEdit	SmallInt	System bitmap
BitmapViewBreak	SmallInt	System bitmap
BitmapViewCallStack	SmallInt	System bitmap
BitmapViewDebugger	SmallInt	System bitmap
BitmapViewMethods	SmallInt	System bitmap
BitmapViewSource	SmallInt	System bitmap
BitmapViewTracer	SmallInt	System bitmap
BitmapViewTypes	SmallInt	System bitmap
BitmapViewWatch	SmallInt	System bitmap
BitmapVSpacing	SmallInt	System bitmap
BitmapXtabTool	SmallInt	System bitmap

{button ,AL(`OPAL_CONST_CONSTYPES;',0,"Defaultoverview",)} **Related Topics**

ToolbarButtonType constants

Constant	Data type	Description
ToolbarButtonPush	SmallInt	Specifies a pushbutton type toolbar button
ToolbarButtonRadio	SmallInt	Specifies a radiobutton type toolbar button
ToolbarButtonRepeat	SmallInt	Specifies a repeating pushbutton type toolbar button
ToolbarButtonToggle	SmallInt	Specifies a toggle-action toolbar button

{button ,AL(`OPAL_CONST_CONSTYPES;',0,"Defaultoverview",)} Related Topics

ToolbarClusterID constants

A cluster is a logical aggregation of buttons There are 13 clusters in the system. Each cluster is always at the same position. The position of the cluster is expressed in 'Button widths' on a horizontal Toolbar. For example, the Mode cluster starts at a distance of 4 button widths from the left.

Constant	Data type	Description
ToolbarFileCluster	SmallInt	Specifies Toolbar cluster 0 (position 0)
ToolbarEditCluster	SmallInt	Specifies Toolbar cluster 1 (position 1, 2, 3)
ToolbarModeCluster	SmallInt	Specifies Toolbar cluster 2 (position 4, 5)
ToolbarToolCluster	SmallInt	Specifies Toolbar cluster 3 (position 6, 7)
ToolbarVCRCluster	SmallInt	Specifies Toolbar cluster 4 (position 8, 9)
ToolbarInterCluster	SmallInt	Specifies Toolbar cluster 5 (position 10, 11, 12, 13)
ToolbarInter2Cluster	SmallInt	Specifies Toolbar cluster 6 (position 14)
ToolbarQuickCluster	SmallInt	Specifies Toolbar cluster 7 (position 15, 16, 17)
ToolbarMiscCluster	SmallInt	Specifies Toolbar cluster 8 (position 18, 19)
ToolbarMisc2Cluster	SmallInt	Specifies Toolbar cluster 9 (position 20, 21)
ToolbarObjectCluster	SmallInt	Specifies Toolbar cluster 10 (position 22)
ToolbarProjectCluster	SmallInt	Specifies Toolbar cluster 11 (position 23)
ToolbarExpertCluster	SmallInt	Specifies Toolbar cluster 12 (position 24)

{button ,AL(`OPAL_CONST_CONSTYPES;',0,"Defaultoverview",)} **Related Topics**

ToolbarState constants

Constant	Data type	Description
ToolbarStateBottom	SmallInt	Specifies a Toolbar docked at screen bottom
ToolbarStateFloatHorizontal	SmallInt	Specifies a floating horizontal Toolbar
ToolbarStateFloatVertical	SmallInt	Specifies a floating vertical Toolbar
ToolbarStateLeft	SmallInt	Specifies a Toolbar docked at screen left
ToolbarStateRight	SmallInt	Specifies a Toolbar docked at screen right
ToolbarStateTop	SmallInt	Specifies a Toolbar docked at screen top

{button ,AL(`OPAL_CONST_CONSTYPES;',0,"Defaultoverview",)} **Related Topics**

TrackBarStyles constants

Constant	Data type	Description
LineSize	SmallInt	The number of ticks the thumb moves on LineUp and LineDown events
PageSize	SmallInt	The number of ticks the thumb moves on PageUp and PageDown events
TrackBarAutoTic	Logical	Automatically displays tick marks
TrackBarBoth	Logical	Displays ticks on both sides of the trackbar
TrackBarBottom	Logical	Thumb points down, tick marks at bottom (TrackBarHorz only)
TrackBarEnableSelRange	Logical	Enables a selected range within the trackbar, used with a SelStart and SelEnd value, highlights the selection range
TrackBarHorz	Logical	Displays trackbar horizontally
TrackBarLeft	Logical	Thumb, tick marks on left-hand side of trackbar (TrackBarVert only)
TrackBarNoTics	Logical	Do not display tick marks
TrackBarRight	Logical	Thumb, tickmarks on right-hand side of trackbar (TrackBarVert only)
TrackBarTop	Logical	Thumb points up, tick marks at top (TrackBarHorz only)
TrackBarVert	Logical	Displays trackbar vertically

{button ,AL(`OPAL_CONST_CONSTYPES;',0,"Defaultoverview",)} **Related Topics**

UIObjectTypes constants

Constant	Data type	Description
BandTool (5.0)	SmallInt	Creates a report band
BoxTool	SmallInt	Creates a box
ButtonTool	SmallInt	Creates a button
ChartTool	SmallInt	Creates a chart
EllipseTool	SmallInt	Creates an ellipse
FieldTool	SmallInt	Creates a field
GraphicTool	SmallInt	Creates a graphic object
LineTool	SmallInt	Creates a line
OleTool	SmallInt	Creates an OLE object
NoteBookTool (7)	SmallInt	Creates a tabbed notebook object
PageBrkTool (5.0)	SmallInt	Creates a page break in a report
RecordTool	SmallInt	Creates a record
TableFrameTool	SmallInt	Creates a table frame
TextTool	SmallInt	Creates a text box
XtabTool	SmallInt	Creates a crosstab object

{button ,AL(`OPAL_CONST_CONSTYPES;',0,"Defaultoverview",)} Related Topics

ValueReasons constants

Constant	Data type	Description
EditValue	SmallInt	The built-in **newValue** method of a radio button field, list, or drop-down edit list has been triggered (e.g., by choosing a radio button or list item), but the field value has not been committed (e.g., by moving off the field).
FieldValue	SmallInt	A field's built-in **newValue** method has been triggered, and the value has been committed.
StartupValue	SmallInt	A field's built-in **newValue** method has been triggered because the form has opened.

{button ,AL(`OPAL_CONST_CONSTYPES;',0,"Defaultoverview",)} **Related Topics**

WindowStyles constants

Constant	Data type	Description
WinDefaultCoordinate	LongInt	Displays a window at its default size and position
WinStyleBorder	LongInt	Specifies a sizing border
WinStyleControlMenu	LongInt	Specifies a system-control menu
WinStyleDefault	LongInt	Specifies default displays attributes
WinStyleDialog	LongInt	Specifies dialog box attributes
WinStyleDialogFrame	LongInt	Specifies a dialog box frame
WinStyleHScroll	LongInt	Specifies a horizontal scroll bar
WinStyleHidden	LongInt	Makes a window invisible
WinStyleMaximize	LongInt	Displays a window at full size
WinStyleMaximizeButton	LongInt	Specifies a maximize button
WinStyleMinimize	LongInt	Displays a window as an icon (minimized)
WinStyleMinimizeButton	LongInt	Specifies a minimize button
WinStyleModal	LongInt	Makes a window modal
WinStyleThickFrame	LongInt	Specifies a thick frame
WinStyleTitleBar	LongInt	Specifies a Title Bar
WinStyleVScroll	LongInt	Specifies a vertical scroll bar

{button ,AL(`OPAL_CONST_CONSTYPES;',0,"Defaultoverview",)} Related Topics

Basic language elements

You can use basic language elements to assign values, call functions from dynamic link libraries (DLLs), and to build control structures like **if...then...else...endIf** loops, **while...endWhile** loops, and **switch...case...endSwitch** structures. You can also use the basic language elements to declare methods, procedures, constants, variables, and data types. Most of these elements are not bound to specific object types; they work for all object types.

Click on a language element for more information.

; (comments)	method
{} (comments)	passEvent
= (equals)	proc
= (assignment)	quitLoop
const	return
disableDefault	scan
doDefault	switch
enableDefault	try
for	type
forEach	uses
if	var
iif	while
loop	

{button ,AL(`BLE_OVERVIEW;INTRO_COMPONENTS;INTRO_INTRO;',0,"Defaultoverview",)} **Related Topics**

Print related ObjectPAL methods and examples

; (comments) keyword

Designates the beginning of a comment, which is text that is ignored by the compiler. The comment extends from the comment operator (;) to the end of the current line.

Syntax

```
; Comments
```

Description

Comments are useful for documenting code.

Example

{button ,AL(`BLE_OVERVIEW;BLE_COMMENT;',0,"Defaultoverview",)} **Related Topics**

; (comments) example

The following example demonstrates the comment operator (;):

```
var
   x AnyType; declares the variable x of AnyType
endvar
x = 25      ; x gets a value of 25
; Comments that begin with the comment operator (;) extend only to
; the end of the current line.
```

{ } (comments) keyword

Designates a comment, which is text that is ignored by the compiler. The comment extends from the open brace ({) to the closing brace (}); it does not end at the end of the line.

Syntax

{*Comments ...*

*...More Comments***}**

Description

Comments are used to document code.

Example

{button ,AL(`BLE_OVERVIEW;BLE_COMMENT;',0,"Defaultoverview",)} Related Topics

{ } (comments) example

The following example demonstrates the comment braces { } operator:

```
var
   x AnyType {declares the variable x of AnyType}
endvar
x = 25       {x gets a value of 25}
{Comments that begin with the comment braces operator extend from the opening brace to the
closing brace, regardless of the number of lines occupied.}
```

= (Assignment operator & Comparison operator) keyword

Syntax

itemSpec **=** *expression*

Description

Normally, in an expression, the = is a comparison operator that tests whether the two operands are equal. Otherwise, the **=** operator assigns the value of *expression* to *itemSpec*. Any previous value stored in *itemSpec* is lost. When assigning a value to an object, information in *itemSpec* can include the containership path.

When you use = with numbers, you can assign a numeric value to a field or variable. For example, the following code assigns the value 1.5 to *i.*

```
i=1.5
```

You can also use Hex values, like those used in C++ or Borland Delphi, to make numeric assignments. The following lines of code assigns 11 to *i*.

```
i = 0x0B

i = 11
```

When you use = with UIObjects, you assign the value of one UIObject to another UIObject. For example, suppose a form contains two fields, *fieldOne* and *fieldTwo*. The following statement copies the value of *fieldTwo* into *fieldOne*.

```
fieldOne = fieldTwo ; fieldOne gets the value of fieldTwo
```

You can also use = with UIObject variables. ObjectPAL uses **attach** the way C and Pascal use pointers. For example,

```
var ui UIObject endVar
ui.attach(fieldOne) ; tells ui to "point to" fieldOne
ui.view() ; displays the value of ui (same as fieldOne) in a dialog box.
ui = fieldTwo ; ui gets the value of fieldTwo (fieldOne value changes, too)
ui.view() ; displays the value of ui (same as fieldTwo) in a dialog box
ui.color = Red ; sets the color of ui and therefore of fieldOne to red
```

The following statement assigns to *ui* all of the information about *fieldOne*:

```
ui.attach(fieldOne)
```

In contrast, the following statement assigns to *ui* (and to *fieldOne*) only the value of *fieldTwo*:

```
ui = fieldTwo
```

Example
{button ,AL(`BLE_OVERVIEW;BLE_EQUAL;',0,"Defaultoverview",)} **Related Topics**

= example

The following example shows various uses of = both as a comparison operator and as an assignment. MyTable is a table frame or multi-record object on the form which contains the fields myField and fieldOne. bigBox, bigCircle, smallBox, and smallCircle are UI Objects on the form which are contained within each other. amountField is a field UI Object on the form.

```
method pushButton(var eventInfo Event)

var
   x     AnyType
   ar    Array[5] AnyType
   w     Logical
   y, z  SmallInt
   tempAmountField   Number
   fred, sam         UIObject
endVar
x = 5.14                ; x gets a value of 5.14 (the data type is Number)
ar[1] = "Hello"         ; element 1 of ar gets the value of "Hello" (String)
y = 5                   ; y gets the value of 5
z = 12                  ; z gets the value of 12
x ="foo"                ; x gets a new value: the String "foo"
myTable.myField = y + z ; the field myField gets the value of y + z
amountField = tempAmountField
bigBox.bigCircle.smallBox.smallCircle.color = Blue
; the color property of smallCircle gets the value of Blue
; the first = assigns a value, all others compare
w = (y = z) ; w gets a value of True if y = z,
            ; otherwise, w gets a value of False
fred.attach(fieldOne) ; makes fred a "pointer" to fieldOne
sam = fred ; assigns the value of fred to sam
```

const keyword

Declares constants.

Syntax

```
const
   constName = { dataType ( value )|value }
endConst
```

Description

const declares one or more constant values, where *dataType*, if included, specifies the data type of the constant. If *dataType* is omitted, the data type is inferred from *value* as either a LongInt, a Number, a SmallInt, or a String.

 **Note**

- You declare constants in a **const...endConst** block in ObjectPAL code or in the Const window in the Object Explorer.

Example

{button ,AL(`BLE_OVERVIEW;BLE_DECLARATION;',0,"Defaultoverview",)} Related Topics

const example

The following example demonstrates how const declares a value.

```
const
   a = -1000                          ; SmallInt, inferred
   x = 123.45                         ; Number, inferred
   newYear = Date ("01/01/99")        ; Date, assigned
   companyName = String ("Corel")  ; String, assigned
endconst
```

disableDefault keyword

Disables the default code for a built-in event method.

Syntax

```
disableDefault
```

Description

disableDefault prevents an event's built-in code from executing for the current call to a built-in event method. Normally, the built-in code executes implicitly at the end of a method, just before the **endMethod** keyword. Using **disableDefault** in a method disables the implicit call to the built-in code.

Example

{button ,AL(`BLE_OVERVIEW;BLE_DEFAULT;',0,"Defaultoverview",)} Related Topics

disableDefault example

The following example sets the value of a field to "hello" when the user types a character. The call to **disableDefault** prevents the built-in code from executing, so the character does not display in the field. The **message** statement displays the character in the Status Bar.

```
method keyChar(var eventInfo KeyEvent)
   self.value = "hello"      ; hello appears in the field
   disableDefault            ; disable the built-in code
   message(eventInfo.char()) ; displays the character in the status bar
endMethod
```

doDefault keyword

Executes the default code for a built-in event method.

Syntax

```
doDefault
```

Description

doDefault executes the built-in code for an event immediately, instead of at the end of the method. Using **doDefault** in a method disables the implicit call to the built-in code. If a method contains more than one **doDefault** statement, only the first one executes; the other statements are ignored.

Generally, if you attach code to an object's built-in **open** method, you should call **doDefault** before calling any other method or procedure. The call to **doDefault** executes the built-in code, ensuring the object is completely opened and initialized.

Examples

{button ,AL(`BLE_OVERVIEW;BLE_DEFAULT;',0,"Defaultoverview",)} **Related Topics**

doDefault examples

Example1	Effects of a call to **doDefault**
Example2	Making a call to **doDefault**

doDefault example 1

The following example demonstrates the effect of a call to **doDefault**. In the following method, the button pushes in, waits two seconds and then the system beeps and the button pops out. The built-in code is called implicitly, just before the **endMethod** statement:

```
method pushButton(var eventInfo Event)
   sleep(2000)
   beep()
endMethod
```

In the following method, the call to **doDefault** makes the button pop out before it sleeps and beeps, and it disables the implicit code at the end of the method.

```
method pushButton(var eventInfo Event)
   doDefault
   sleep(2000)
   beep()
endMethod
```

doDefault example 2

The following example shows how to call **doDefault** when you attach code to an object's built-in **open** method. The following code is attached to the built-in **open** method of an unbound field object named *greetingFld*. The code calls **doDefault** to execute the built-in code and then sets the value of the field object.

```
greetingFld::open
method open(var eventInfo Event)
   doDefault
   self.Value = "Hello " + getNetUserName()
endMethod
```

enableDefault keyword

Enables the default code for a built-in event method.

Syntax

```
enableDefault
```

Description

enableDefault allows the built-in code to execute normally at the end of a method, just before the **endMethod** statement. Compare **enableDefault** to **doDefault**, which executes the built-in code immediately.

Example

{button ,AL(`BLE_OVERVIEW;BLE_DEFAULT;',0,"Defaultoverview",)} Related Topics

enableDefault example

In the following example, default behavior is disabled and custom methods **doOpen()** or **doQuit()** are called if the respective conditions apply. Otherwise, the default behavior is enabled.

```
method menuAction(var eventInfo MenuEvent)


var theChoice String endVar
disableDefault
theChoice = eventInfo.menuChoice()
switch
   case theChoice = "Open" : doOpen()
   case theChoice = "Quit" : doQuit()
   otherwise               : enableDefault
endSwitch
endMethod
```

for keyword

Executes a sequence of statements a specified number of times.

Syntax

```
for counter [ from startVal  [ to endVal ] [ step stepVal ]
      Statements
endFor
```

Description

for executes a sequence of Statements as many times as is specified by a counter, which is stored in *counter* and controlled by the optional **from**, **to**, and **step** keywords. Any combination of these keywords can be used to specify the number of times the statements in the loop are executed. You don't have to declare *counter* explicitly, but a **for** loop runs faster if you do.

The arguments *startVal*, *endVal*, and *stepVal* are values or expressions representing the beginning counter value, ending counter value, and the number by which to increment the counter each time through the loop. These values can be any data type represented by AnyType, except Point, Memo, Graphic, String, OLE, and Binary. Also, *counter* must be a literal value or a single-valued variable; it can't be an array element or record field value.

You can use **for** without the **from**, **to**, and **step** keywords:

- If *startVal* is omitted, the counter starts at the current value of *counter*.
- If *endVal* is omitted, the **for** loop executes indefinitely.
- If *stepVal* is omitted, the counter increments by 1 each time through the loop.
- *startVal*, *endVal*, and *stepVal* are stored in a temporary buffer; they are not evaluated each time through the loop.

If **quitLoop** is used within the body of statements in the **for** loop, the **for ...endFor** loop is exited. If **loop** is used within the body of statements, statements following **loop** are skipped, the counter is incremented, and iteration continues from the top of the **for** loop.

If **step** is positive and a **to** clause is present, iteration continues as long as the value of *counter* is less than or equal to the value of *endVal*. If **step** is negative, iteration continues as long as the value of *counter* is greater than or equal to the value of *endVal*. In either case, when the value of *counter* reaches or exceeds the limit set by **step**, the **for** loop stops executing, but *counter* keeps its value, as shown in the example.

If *counter* has not previously been assigned a value, **from** creates the variable and assigns to it the value of *startVal*.

Example

{button ,AL(`BLE_OVERVIEW;BLE_CONTROL_STRUCTURE;',0,"Defaultoverview",)} Related Topics

for example

The following example demonstrates a simple **for** loop. Notice the value of the counter variable *i* after the **for** loop is completed.

```
var i SmallInt endVar
for i from 1 to 3
   i.view("Inside for loop") ; i = 1, i = 2, i = 3
endFor
i.view("Outside for loop")   ; i = 4
```

forEach keyword

Repeats the specified statement sequence in elements within a dynamic array, or DynArray.

Syntax

```
forEach VarName in DynArrayName
      Statements
endForEach
```

Description

forEach walks through the elements in a DynArray. The argument *VarName* is a String variable used as a placeholder for the DynArray indexes. The argument *DynArrayName* is a DynArray variable that identifies the DynArray to walk through. If *DynArrayName* does not exist, the **forEach** statement causes an error when the method is compiled. The *Statements* clause represents the one or more ObjectPAL statements that are to be executed for each index in the DynArray.

Generally, you cannot use the **for** statement to step through a DynArray because the indexes of a DynArray are not necessarily integers. Because DynArray indexes are not integers, DynArray elements are not ordered sequentially. The **forEach** statement operates on DynArray elements in an arbitrary order. You should not rely on a specific ordering of indexes.

If the **quitLoop** statement is used within the body of statements in the **forEach** loop, the **forEach...endForEach** loop is exited. If the loop statement is used within the body of *Statements*, the statements following **loop** are skipped and iteration continues from the top of the **forEach** loop.

Do not call **removeItem** or **empty** to modify a DynArray in a **forEach** loop.

Example

{button ,AL(`BLE_OVERVIEW;BLE_CONTROL_STRUCTURE;',0,"Defaultoverview",)} Related Topics

forEach example

The following example uses the **forEach** statement to display the elements in the dynamic array, or DynArray, created by the **sysInfo** statement:

```
var
   SystemArray DynArray[] AnyType
   Element AnyType
endVar
sysInfo(SystemArray)
forEach Element IN SystemArray
   message(Element, " : ", SystemArray[Element])
   sleep(1500)
endForEach
```

if keyword

Executes one of two sequences of statements, depending on the value of a logical condition.

Syntax

```
if Condition then
      Statements1
[else
      Statements2 ]
endIf
```

Description

When ObjectPAL comes to an **if** statement, it evaluates whether the *Condition* is True. If so, it executes the statements listed in *Statements1* in sequence. If not, it skips *Statements1* and, if the optional **else** keyword is present, executes the statements in *Statements2*. In either case, execution continues after the **endIf** keyword.

An **if** construction can span several lines, especially if there are many statements in *Statements1* or *Statements2*. It is recommended that you indent the **then** and **else** clauses to show the flow of control.

```
if Condition then
   Statements1
else
   Statements2
endIf
```

```
The following is an example of an if statement:
if Stock < 100 then
   AddStock()        ; execute a custom method called AddStock()
   Stock = Stock + 10 ; then, add 10 to the value of Stock
endIf
```

if statements can be nested; that is, any of the statements in *Statements1* or *Statements2* can also be **if** statements. Nested **if** statements must be fully contained within the controlling **if** structure, in other words, each nested **if** statement must have an **endIf** within the nest. As in the following code, each **if...endIf** set must enclose code or code and another complete **if...endIf set**.

```
if Condition then
   if Condition then
      Condition
   endIf
endIf
```

Example

{button ,AL(`BLE_OVERVIEW;BLE_CONTROL_STRUCTURE;',0,"Defaultoverview",)} **Related Topics**

if example

The following example provides code for a nested **if** statement:

```
if skillLevel = "Beginner" then
   if skillBox.color = "Red" or skillBox.color = "Yellow" then
      skillBox.color = "Green"
   endIf
endIf
```

iif keyword

Returns one of two values, depending on the value of a logical condition.

Syntax

iif (*Condition*, *ValueIfTrue*, *ValueIfFalse* **)**

Description

iif (immediate **if**) allows branching within a single statement. You can use **iif** anywhere you use other expressions. **iif** is especially useful in calculated fields on forms or reports where **if...endIf** statements are illegal.

Example

{button ,AL(`BLE_OVERVIEW;BLE_CONTROL_STRUCTURE;',0,"Defaultoverview",)} Related Topics

iif example

The following example demonstrates how an **iif** keyword returns a value.

```
a = iif(x > 1, b, c) ; if x > 1, a = b; else a = c
```

loop keyword

Passes control to the top of the nearest enclosing **for**, **forEach**, **scan**, or **while** loop.

Syntax

```
loop
```

Description

When executed within a **for**, **forEach**, **scan**, or **while** structure, **loop** skips the statements between it and the **endFor**, **endForEach**, **endScan**, or **endWhile** loops and returns to the beginning of the structure. Otherwise, **loop** causes an error.

Example

{button ,AL(`BLE_OVERVIEW;BLE_CONTROL_STRUCTURE;',0,"Defaultoverview",)} **Related Topics**

loop example

The following example shows how the loop keyword passes control to the nearest **for** statement.

```
var x SmallInt endVar


for x from 1
   if x <> 5 then
      loop ; go back to for statement, get next value of x
      message("This never appears") ; this statement never executes
   else
      quitLoop                      ; break out of the loop
   endIf
endFor
message(x) ; displays 5
```

method keyword

Defines an ObjectPAL method.

Syntax

method Name **(** *parameterDesc* [, *parameterDesc*] *] **)** [*returnType*]

[**const** *section*]

[**type** *section*]

[**var** *section*]

Statements

endMethod

Description

method marks the beginning of a method. You must provide the following:

- the method name in *Name*
- parentheses, even if the method has no arguments
- the *Statements* that comprise the method

The definition ends with the mandatory **endMethod** keyword.

Additionally, you can declare constants, data types, variables and procedures before the **method** keyword, and you can declare variables and constants after the **method** keyword.

Also optional are one or more parameter descriptions (up to a maximum of 29) represented in the prototype by *parameterDesc*, where each description takes the following form:

[**var|const**] ***parameter type***

The optional *returnType* declares the data type of the value returned by the method. *returnType* is optional because a method may or may not return a value. However, if the method returns a value, you must specify the data type of the value.

Methods and procedures differ in the following ways:

- Methods are visible and exportable to other objects, while procedures are private within a containership hierarchy.
- A method can contain a procedure definition, but a procedure can't contain a method definition.

Note

- The scope of a method depends on where it is declared.

Example

{button ,AL(`BLE_OVERVIEW;BLE_DECLARATION;',0,"Defaultoverview",)} Related Topics

method example

```
method pushButton (var eventInfo Event)
   var
      txt String
      myNum Number
   endVar
   myNum = 123.321
   txt = String(myNum)
   msgInfo("myNum = ", txt)
endMethod
```

passEvent keyword

Passes the event to the object's container.

Syntax

```
passEvent
```

Description

passEvent passes the event packet to the object's container. Using **passEvent** in a method does not affect the implicit call to the built-in code.

Example

{button ,AL(`BLE_OVERVIEW;BLE_DEFAULT;',0,"Defaultoverview",)} **Related Topics**

passEvent example

The code in the following example is attached to a field object. It executes when the pointer is in the field object. If SHIFT is held down when the mouse is clicked, the code calls **disableDefault** to prevent the built-in code from executing and calls **passEvent** to send the event to the field object's container. This technique is useful when you want several objects to respond the same way to a given event.

```
method mouseDown(var eventInfo MouseEvent)
   if eventInfo.isShiftKeyDown() then
      disableDefault
      passEvent ; let container handle it
   endIf
endMethod
```

proc keyword

Defines an ObjectPAL procedure.

Syntax

```
proc ProcName ( [ parameterDesc [ , parameterDesc ] * ] ) [ returnType ]
[ const section ]
[ type section ]
[ var section ]
Statements
endProc
```

Description

proc begins the definition of a procedure. You must provide the following:

- the procedure name, in *ProcName*
- parentheses, even if the procedure has no arguments
- zero or more parameter descriptions (up to a maximum of 29) represented in the prototype by *parameterDesc*, where each description takes the following form:

  [**var**|**const**] ***parameter type***
- use *returnType* to declare the data type of the value returned by the procedure (if it returns a value)
- sections to declare variables, constants, and types
- the *Statements* that comprise the procedure

The definition ends with the mandatory **endProc** keyword.

You can use **return** in the body of a procedure to return a value to the calling method or procedure.

A procedure used in an expression must return a value, such as

x = NumValidRecs("Orders") ; NumValidRecs is a procedure

Notes

- You declare procedures in a **proc...endProc** block in ObjectPAL code or in the Proc window in the Object Explorer.

Procedures and methods are similar. The key differences are that methods are visible and exportable to other objects, while procedures are private within a containership hierarchy. A method can contain a procedure definition, but a procedure can't contain a method definition.

- The scope of a procedure depends on where it is declared.

Example

{button ,AL(`BLE_OVERVIEW;BLE_DECLARATION;',0,"Defaultoverview",)} Related Topics

proc example

```
proc inc (x SmallInt) SmallInt
   return x + 1
endProc
method pushButton(var eventInfo Event)
   var x SmallInt endVar
   x = 5
   x = inc(x) ; calls the procedure
   message(x) ; displays 6
endMethod
```

quitLoop keyword

Terminates the **for**, **forEach**, **scan**, or **while** loop in which it appears.

Syntax

```
quitLoop
```

Description

quitLoop exits immediately from the closest enclosing **for**, **forEach**, **scan**, or **while** loop. The method continues with the statement following the closest **endFor**, **endForEach**, **endScan**, or **endWhile**.

quitLoop causes an error if executed outside of a **for**, **scan**, or **while** structure.

Example

{button ,AL(`BLE_OVERVIEW;BLE_CONTROL_STRUCTURE;',0,"Defaultoverview",)} **Related Topics**

quitLoop example

In the following example, **quitLoop** is used in a **for** loop to determine whether an array has any unassigned elements:

```
var
   myArray Array[12]
   notAssigned Logical
endVar
notAssigned = False
for i from 1 to 12
   if not isAssigned(myArray[i]) then
      notAssigned = True
      quitLoop
   endIf
endFor
```

return keyword

Returns control to a method or procedure, optionally passing back a value.

Syntax

return [*Expression*]

Description

return is used to return control from the current procedure or method to the procedure or method that called it, whether or not the method or procedure is declared to return a value. The following rules apply to **return**:

- If **return** is executed within the body of a procedure, the procedure is exited.
- If **return** is executed within a method (but outside of the body of a procedure), the method is exited.

You can optionally return the value of *Expression* when returning from either a procedure or a method. If a procedure is called in an expression, then the procedure must return a value, which becomes the value of the procedure call.

```
y = myProc(x) + 3      ; myProc is a procedure
```

If a procedure is called in a standalone context, then any returned value is ignored. For example:

```
myProc(x)
```

If no *Expression* is supplied, **return** must not be followed by anything else on the line other than a comment.

The following data types *cannot* be returned: DDE, Database, Query, Session, Table, or TCursor.

It is not necessary to use **return** to pass control back to a higher-level method or procedure, since this happens automatically when a lower-level method or procedure finishes. However, if the method or procedure is declared to return a value, you must use **return** to return the value; the value won't be returned automatically.

Example

{button ,AL(`BLE_OVERVIEW;BLE_CONTROL_STRUCTURE;',0,"Defaultoverview",)} Related Topics

return example

The following example adds one to the value of a variable and returns the new value to the calling method:

```
proc addOne (x SmallInt) SmallInt
   return x + 1
endProc
```

In a built-in event method, a **return** statement executes the built-in code unless you explicitly disable the code. For example, the following code calls **return** when the user types a ? into a field object. The call to **disableDefault** prevents the built-in code from displaying the ? in the field object.

```
method keyChar(var eventInfo KeyEvent)
   if eventInfo.char() = "?" then
      disableDefault
      return
   endIf
endMethod
```

scan keyword

Scans the TCursor and executes ObjectPAL instructions.

Syntax

```
scan tcVar [ for booleanExpression ] :
      Statements
endScan
```

The colon is required, even if you omit the **for** keyword.

Description

scan searches *tcVar* (TCursor) and executes *Statements* (ObjectPAL instructions) for each record. **scan** always begins at the first record of the table and examines each record in sequence. When statements in the **scan** loop change an indexed field, that record moves to its sorted position in the table; It's possible, therefore, to encounter the same record more than once in the same loop.

If you supply the **for** clause, *Statements* execute only for those records that satisfy the condition; all other records are skipped. If the table is empty or if no records meet the condition, the **scan** has no effect.

scan allows you to prototype a statement sequence for a single record of a table and then place that sequence inside a **scan** loop to apply it to an entire table.

You can use **loop**, **return**, and **quitLoop** in the body of the **scan**. **loop** skips the remaining statements between it, and **endScan**, moves to the next record, and returns to the top of the **scan** loop. **quitLoop** terminates the **scan** altogether, leaving the record being scanned as the active record.

Since **scan** repeats an entire statement sequence for each record, don't include actions that only need to be performed once for the table. Put those statements outside the **scan** loop. **scan** automatically moves from record to record through the table, so there's no need to call **nextRecord**.


Example

{button ,AL(`BLE_OVERVIEW;BLE_CONTROL_STRUCTURE;',0,"Defaultoverview",)} Related Topics

scan example

The following example uses a **scan** loop to update the *Employee* table. It scans the Dept. field of each record, and if the value is Personnel, changes it to Human Resources.

```
var
    empTC TCursor
endVar


empTC.open("employee.db") ; These statements need only be executed once,
empTC.edit()              ; so they're placed outside the loop.


scan empTC for empTC.Dept = "Personnel":  ; the colon is required
   empTC.Dept = "Human Resources"
endScan


empTC.endEdit()
empTC.close()
```

switch keyword

Executes a specified set of statement sequences.

Syntax

```
switch
   CaseList
   [ otherWise: Statements ]
endSwitch
```

CaseList is any number of statements in the following form:

```
case Condition : Statements
```

Description

switch uses the values of the *Condition* statements in *CaseList* to determine which sequence of *Statements* should be executed, if any. **switch** works like multiple **if** statements, and each *CaseList* works like a single **if** statement.

The case *Conditions* are evaluated in the order in which they appear:

- if one has a value of True, the corresponding *Statements* sequence is executed and the rest are skipped
- if none has the value True and the optional **otherWise** clause is present, the *Statements* in **otherWise** are executed
- if none has the value True and no **otherWise** clause is present, **switch** has no effect

Thus, one set of *Statements* is executed at most. The method resumes with the next statement after **endSwitch**.

Example

{button ,AL(`BLE_OVERVIEW;BLE_CONTROL_STRUCTURE;',0,"Defaultoverview",)} Related Topics

switch example

The following example creates an array of 100 random numbers and then uses the bubble sort algorithm to sort the numbers in numerical order:

```
method pushButton(var eventInfo Event)
var
   sz, i , itmp, j,k SmallInt
   a Array[100] SmallInt
   tmp Number
endVar


   sz = 100
   a.fill(0)

for i from 1 to sz step 1
   tmp = Rand()
   switch
         case tmp < .1 : a[i] = 1
         case tmp < .2 : a[i] = 2
         case tmp < .3 : a[i] = 3
         case tmp < .4 : a[i] = 4
         case tmp < .5 : a[i] = 5
         case tmp < .6 : a[i] = 6
         case tmp < .7 : a[i] = 7
         case tmp < .8 : a[i] = 8
         case tmp < .9 : a[i] = 9
         otherwise:      a[i] = 10
   endSwitch
endFor


for i from 1 to sz-1 step 1
   for j from 1 to sz-i step 1
      if a[j] <> a[j+1] then
         a.exchange(j, j+1)
      endIf
   endFor
endFor


endMethod
```

try keyword

Marks a block of statements to try, and specifies a response should an error occur.

Syntax

```
try
   [ Statements ] ; the transaction block
onFail
   [ Statements ] ; the recovery block
   [ reTry ]          ; optional
EndTry
```

Description

The **try...onFail** block builds error recovery into an application is .

The transaction block is a set of *Statements*. If the transaction block succeeds, the program skips to **endTry**. If the transaction fails, the recovery block executes. You can call **reTry** to execute the transaction block again.

The program calling the System procedure **fail** causes a trial to fail by at some point within the transaction block or within procedures called by the transaction block. This stops system functions from returning status errors or null values to their callers.

A **fail** call can be nested within several procedure calls. Their local variables are removed from the stack, and any special objects (such as large text blocks) are deallocated. If reference objects (such as tables) are in use, they are closed, and any pending updates are canceled. It's as if the transaction had never started. What remains are changes to variables outside of the block, or data added successfully to tables and committed before the failure occurred.

If during a recovery block you decide that the error code is not one you expected or is more serious than can be handled at this level, call **fail** again to pass that error code. If no higher-level **try...onFail** block exists, the whole application fails, existing actions are canceled, and resources are closed.

By default, a **try...onFail** block traps critical errors only. Use **errorTrapOnWarnings** if you want a **try...onFail** block to also trap warnings.

Example

{button ,AL(`BLE_OVERVIEW;BLE_ERROR;',0,"Defaultoverview",)} Related Topics

try example

The following example attempts to set the Color property of some design objects and uses a **try...onFail** block to handle the situation if the property cannot be set.

```
method pushButton(var eventInfo Event)
var s String endVar
box1.box2.color = Blue                    ; this works
s = "box5"                                ; box5 doesn't exist


try
   box1.(s).color = Red                   ; try to set color of box5
onFail                                    ; handle the error
   msgStop("Error", "Couldn't find " + s)
   s = "box2"                             ; box2 exists
   reTry                                  ; try again
endTry


s = "box6"                                ; box6 doesn't exist
try
   box1.(s).color = Green
onFail
   fail(peObjectNotFound, "The object " + s + "does not exist.")
endTry
endMethod
```

type keyword

Declares data types.

Syntax

type

[*newTypeName = existingType*] *

endType

Description

Using **type**, you can define new data types (based on existing ObjectPAL types). Once defined, you can use these types to declare variables in methods.

You declare data types in a **type...endType** block in ObjectPAL code, or in the Type window on the Methods page of the Object Explorer.

For example, an application to track the number of parts in a warehouse might declare a **type** *partQuantity* and then declare a variable to be of **type** *partQuantity*, like this:

```
type
   partQuantity = SmallInt ; declare a new type
endType


var                  ; use the new type to declare a variable
   pQty partQuantity ; pQty is a SmallInt
endVar               ; because partQuantity is a SmallInt
```

Later, if the number of parts approaches 32,767 (the maximum value of a SmallInt), you need only change the **type** definition, for example,

```
type
   partQuantity = LongInt ; change the declaration
endType


var                  ; use the new type to declare a variable
   pQty partQuantity ; pQty is now a LongInt
endVar               ; because partQuantity is a LongInt
```

Example

{button ,AL(`BLE_OVERVIEW;BLE_DECLARATION;',0,"Defaultoverview",)} Related Topics

type example

The following example declares a **record** *Employee* that you can use to declare variables in methods and procedures. Records defined in an object's Type window have no connection to tables. Instead, they are similar to records in Pascal and STRUCTs in C, because they allow you to join several related elements of data together under one name.

```
type
   Employee = record
              LastName   String
              FirstName  String
              Title      String
              Salary     Currency
              DateHired  Date
              endRecord
endType
```

uses keyword

Declares external ObjectPAL methods, types, constants, or dynamic link library (DLL) routines to use in a method or procedure.

Syntax

```
uses ObjectPAL
   [ "fileName"]*
endUses
```

Syntax for declaring DLL routines:

```
uses LibraryName
   [ routineName ( [parameterList] ) [returnType] [[callingConvention ["linkName"]] ]*
enduses
```

Note

- While the syntax shown above is different from the **uses** block syntax in version 5.0, any existing **uses** blocks will continue to work as before.

Description

The **uses** block, declared in an object's Uses window, makes methods, constants, and type definitions available to the object's methods and procedures. An ObjectPAL uses block is different from a DLL uses block, which is why they are discussed separately. A Uses window may contain multiple ObjectPAL or DLL **uses** blocks.

Changes to uses keyword

The **uses** keyword can now be used to specify types, methods and constants from an ObjectPAL form or library. You can use all of the types, methods, and constants in a specific library by specifying the filename of the form or library. You don't have to separately name each of the types, constants, and methods you want to use.

The syntax for specifying a DLL in a **uses** block now includes an optional calling convention that lets you control the type of call made to the DLL.

Note that Corel Paradox for Windows 95 and Windows NT requires 32-bit DLL's. Any DLL compiled for 16-bit use (such as with Windows 3.1) will no longer work.

{button ,AL(`BLE_OVERVIEW;BLE_USES;',0,"Defaultoverview",)} Related Topics

ObjectPAL uses block

To use methods, constants, or type definitions stored in an ObjectPAL library or attached to a form, write a **uses** block in an object's Uses window.

Syntax

```
uses ObjectPAL
   ["fileName"]*
endUses
```

Description

The keyword **ObjectPAL** indicates that you are referencing ObjectPAL forms or libraries rather than a dynamic link library (DLL).

Specify the filename of each form or library name to reference. You may use an alias or path in each specified filename. Each filename must be surrounded by quotation marks and must include the file extension .FSL or .LSL. Each form or library that you reference must be in the .FSL or .LSL format when the **uses** block is compiled.

You must open a form or library before calling a method from it; however, you can use constants and type definitions without opening the form or library.

Every form or library that you want to reference must be explicitly named in the **uses** block. You cannot, for example, have a form FORM1.FSL, with a **uses** block that references LIBRARY1.LSL, that in turn has a **uses** block that references LIBRARY2.LSL, and then use the constants, types, or method declarations defined in LIBRARY2.LSL in the code in FORM1.FSL. (In this case, you would add the **uses** block for FORM1.FSL shown below to use the constants, types, and methods from both LIBRARY1.LSL and LIBRARY2.LSL).

```
Uses ObjectPAL
   "LIBRARY1.LSL" "LIBRARY2.LSL"
endUses
```

Constants and type definitions defined in the **const** and **type** sections of a library are available for other forms, libraries, or scripts to access through a **uses** statement. All methods defined in a library are available after a library variable has been attached to the library containing the methods.

Constants and type definitions defined in the **const** and **type** sections at the *form level only* are available for other forms, libraries, or scripts to access through a **uses** statement. All methods defined on all objects of a form are available to be called after a form variable has been attached to the form containing the methods.

Procedures and variables in external forms or libraries are not available. If you need to access variables in libraries, use methods in the library to get and set the values of library variables. Then you can call those methods from your forms, libraries, or scripts to share global values.

When your code is compiled or saved, it reads the constants, types, or method declarations from the .FSL or .LSL files named in **uses** blocks. Delivered forms or libraries (.FDL and .LDL files) do not have the information required for this step, so you must have the .FSL or .LSL files available when you make changes to your code.

After you deliver your code, it will run without the .FSL or .LSL files it references. After the code is saved, it will run without the .FSL or .LSL files, as long as you don't make changes to your code.

When you change constant or type information in a form or library that other forms, libraries, or scripts reference, all the forms, libraries, or scripts need to be recompiled to use the changed values. To recompile your code, make sure you have the Show Developer Menus check box enabled in the Developer Preferences dialog box. For each library, or script, open the file in Design mode, click Program, Compile, then File, Save.

Examples

{button ,AL(`BLE_OVERVIEW;BLE_USES;',0,"Defaultoverview",)} Related Topics

OPAL uses block examples

Example1	Calculating interest rates
Example2	Referencing an existing library
Example3	Creating a reference library
Example4	The importance of multiple uses blocks

ObjectPAL uses block example 1

The following example calculates interest rates by referencing an ObjectPAL library. The library, named MATHLIB.LSL, contains the method **calcInterest**, which takes two arguments: *intRate* and *nPeriods*. It returns the interest calculated.

The following code, attached to a button's Uses window, reads the declaration for the **calcInterest** method from MATHLIB.LSL so the button can use it.

```
uses ObjectPAL
   "mathlib.lsl"
endUses
```

The following code, attached to a button's built-in **pushButton** method, opens the library, reads the values of two fields on a tableframe, calls **calcInterest**, and then displays the results.

```
method pushButton(var eventInfo Event)
   var
      mathLib   Library
      iRate     Number
      nPeriods  SmallInt
      interest  Number
   endVar
   if mathLib.open("mathlib.lsl") then
      iRate = mortgage.intRate.value
      nPeriods = mortgage.nYears.value * 12
      interest = mathLib.calcInterest(iRate, nPeriods)
      interest.view("Interest")
   endIf
endMethod
```

In the following example, dot notation specifies where to find the **calcInterest** method. The following statement looks in the library represented by the Library variable *mathLib*.

```
interest = mathLib.calcInterest(iRate, nPeriods)
```

The concept for calling a method attached to another form is the same. Use dot notation to specify the form used to search for the method. The following example assumes that the Form variable *codeForm* has been previously declared, and that the form has been opened and referenced in a **uses** block.

```
returnValue = codeForm.getObjHelp(self.name)
```

Note

- With previous versions of Corel Paradox, the **uses** block was used to declare external methods to call. The declarations are now read directly from the form or library that you are calling. You no longer have to maintain multiple copies of method declarations as they change, and Corel Paradox reports parameter mismatches when you compile your code rather than later as your code is run.

ObjectPAL uses block example 2

The following example references an ObjectPAL library named PARTS.LSL. The example shows how the **uses** block allows you to share constants, type declarations, and method declarations from forms and libraries.

The library PARTS.LSL contains a **const** block, a **type** block, and a method using the constants and type definitions.

```
const
   DefaultPartName = "N/A"
   DefaultPartNumber = "000-00"
   DefaultPricePerUnit = 1.00
endConst


type
   PartRecordType = Record
      PartName      String
      PartNumber    String
      QtyOnHand     LongInt
      QtyOnOrder    LongInt
      PricePerUnit  Currency
   endRecord
endType


method NewPart(var newPartRecord PartRecordType)
   newPartRecord.PartName = DefaultPartName
   newPartRecord.PartNumber = DefaultPartNumber
   newPartRecord.QtyOnHand = 0
   newPartRecord.QtyOnOrder = 0
   newPartRecord.PricePerUnit = DefaultPricePerUnit
endMethod
```

The following code, attached to a button's Uses window, declares *DefaultPartName*, *DefaultPartNumber*, *DefaultPricePerUnit* and *PartRecordType* from the library and declares *NewPart* so the button can use them:

```
Uses ObjectPAL
   "parts.lsl"
endUses
```

The following code, attached to a button's built-in **pushButton** method, opens the library and calls the method with a *PartRecordType* variable. Note that *PartRecordType* is a type defined in the library and is declared automatically by the **uses** block.

```
method pushButton(var eventInfo Event)
   var
      partsLib    Library
      partRecord  PartRecordType
   endVar


   if partsLib.open("parts") then
      partsLib.newPart(partRecord)
   endIf
endMethod
```

ObjectPAL uses block example 3

The following example references an ObjectPAL library named WINAPI.LSL. The example shows how to create a *Reference Library,* that is, a library that is only accessed at compile time for constant, type, and method declarations. An ObjectPAL Reference Library contains no ObjectPAL code, only definitions.

Certain data structures, constants, and method declarations that you develop in Corel Paradox applications can apply to several projects. The **uses** block allows applications to access centralized libraries that have been created solely for the purpose of defining the types, constants, and method declarations used. Changes to types and constants automatically propagate to all projects referencing the information (after the projects are recompiled to include the change). An ObjectPAL Reference Library is similar to a header file (.H) in the C and C++ programming languages.

The following code is attached to the Uses window in WINAPI.LSL. It declares calls made to the Windows Application Programming Interface (API). These calls should not change, so you should have them defined in a single file that also does not change, where they can be referenced whenever needed.

```
Uses User32
   GetWindowText(hwin CLONG, title CPTR, nMaxLength CLONG) CLONG  [STDCALL "GetWindowTextA"]
   GetActiveWindow() CHANDLE  [STDCALL "GetActiveWindow"]
   MessageBox(hwin CLONG, text CPTR, title CPTR, flags CLONG) CLONG  [STDCALL "MessageBoxA"]
EndUses
```

The following code is attached to the Const window in WINAPI.LSL. It assigns a constant used in the *MessageBox* call to the Windows API.

```
Const
   MB_OK = 0
EndConst
```

The following code, attached to a **pushButton** method, calls the functions from the Windows API defined in WINAPI.LSL:

```
uses ObjectPAL
   "winapi.lsl"
enduses


method pushButton(var eventInfo Event)
var
   windowHandle   LongInt
   windowTitle   String
endvar


   windowTitle = fill(" ", 80)   ; reserve 80 characters for title
   windowHandle = GetActiveWindow()
   if GetWindowText(windowHandle, windowTitle, 80) > 0 then
      MessageBox(0, windowTitle, "Title of Active Window", MB_OK)
   endif


endmethod
```

Other objects (forms, libraries, or scripts) can also access WINAPI.LSL with a **uses** block and declare USER32.DLL as the Windows functions in the system dynamic link library (DLL). It is not necessary to have WINAPI.LSL present at run time in either source (.LSL) or delivered (.LDL) form.

ObjectPAL uses block example 4

The following example references an ObjectPAL library named PARTSHDR.LSL. The example demonstrates how the **uses** block enables you to share constants, type declarations, and method declarations from forms and libraries. It also demonstrates how to use a Reference Library, and that you may need to use multiple **uses** blocks to declare all the information you need.

The library PARTSHDR.LSL contains a **const** block and a **type** block. It defines some global constants and types that are to be used by several other forms and libraries. PARTSHDR.LSL is considered a Reference Library because Corel Paradox only needs to reference the information it contains at compile time.

```
const
   DefaultPartName = "N/A"
   DefaultPartNumber = "000-00"
   DefaultPricePerUnit = 1.00
endConst


type
   PartRecordType = Record
      PartName      String
      PartNumber      String
      QtyOnHand      LongInt
      QtyOnOrder      LongInt
      PricePerUnit   Currency
   endRecord
endType
```

The library PARTS.LSL declares the **NewPart** method. It declares constants and type declarations through a **uses** block that references PARTSHDR.LSL.

```
uses ObjectPAL
   "partshdr.lsl"
endUses


method NewPart(var newPartRecord PartRecordType)
   newPartRecord.PartName = DefaultPartName
   newPartRecord.PartNumber = DefaultPartNumber
   newPartRecord.QtyOnHand = 0
   newPartRecord.QtyOnOrder = 0
   newPartRecord.PricePerUnit = DefaultPricePerUnit
endMethod
```

The following code is attached to a button's Uses window. It declares *DefaultPartName*, *DefaultPartNumber*, *DefaultPricePerUnit*, and *PartRecordType* from PARTSHDR.LSL and *NewPart* from PARTS.LSL so the button can use them:

```
Uses ObjectPAL
   "partshdr.lsl" "parts.lsl"
endUses
```

Even though PARTS.LSL has a **uses** block that references PARTSHDR.LSL, the **uses** block for this button must explicitly include the reference to PARTSHDR.LSL. An indirect reference is not sufficient. Every object that needs to declare constants, type definitions, or methods from external forms or libraries must declare the forms or libraries directly in its own **uses** block or have a definition included in the **uses** block of one of its containers.

The following code, attached to a button's built-in **pushButton** method, opens the library and calls the method with a *PartRecordType* variable.

```
method pushButton(var eventInfo Event)
   var
      partsLib   Library
      partRecord   PartRecordType
   endVar


   if partsLib.open("parts") then
      partsLib.newPart(partRecord)
   endIf


   partRecord.view() ; display the record to show the changed values


endMethod
```

DLL uses block

To use routines stored in a dynamic link library (DLL), write a DLL **uses** block in one of the following places:

- a design object's Uses window
- a window for a built-in method
- a window for a custom method
- a window for a custom procedure

Where you write the block depends on the desired scope (availability) of the routine.

No matter where you write it, the basic structure (shown in the following example) is the same:

Syntax

```
uses libraryName
[ routineName ( [parameterList] ) [returnType] [[callingConvention ["linkName"]] ]*
endUses
```

Description

The required elements in a DLL **uses** block are *libraryName* and an optional list of routines. Each routine must be specified with a *routineName* and the left and right parentheses. All other arguments are optional.

The argument *libraryName* specifies the DLL filename. Corel Paradox assumes a file extension of .DLL or .EXE.

Each routine that you declare must include a *routineName*, the name you use in your ObjectPAL code to call the external routine.

The optional *parameterList* specifies zero or more argument names and data types.

If the routine returns a value, *returnType* specifies the return value's data type.

The *callingConvention* for a DLL call can be PASCAL, STDCALL, or CDECL.

The *linkName* argument is the name of the routine as it is defined in the DLL. It is dependent on the calling convention and is case sensitive in Windows 95 and Windows NT.

Windows searches for the DLL *libraryName* in this order:

1. the current directory
2. the Windows directory (folder). You can use the FileSystem procedure **windowsDir** to find the path to this directory (typically, it's C:\WINDOWS).
3. the Windows system directory (folder). You can use the FileSystem procedure **windowsSystemDir** to get the path to this directory (typically, it's C:\WINDOWS\SYSTEM).
4. the directories listed in the PATH environment variable. Refer to your DOS documentation for more information.
5. the list of directories mapped in a network

Advanced Windows programmers: If you're calling a routine from a previously loaded DLL (e.g., a DLL loaded automatically by Windows), you can use *libraryName* to specify the DLL's module name instead of the filename. Consult your programming language's documentation for more information about DLL module names.

A DLL **uses** block can contain one or more *routineNames*, and each *routineName* can have its own *parameterList*. A *parameterList* specifies zero or more argument names and data types. If the routine returns a value, the *returnType* specifies the return value's data type. ObjectPAL only checks for exact matches in your specifications between these arguments and those arguments declared in the routine.

The routines must fit one of the following descriptions:

- Routines written in assembly language, C, C++, or Pascal and stored in a Windows DLL. A DLL is a library of executable code or data that you can link to your application at runtime. Using DLLs, you can add features and functions without modifying your compiled ObjectPAL application.
- Routines from the Windows API (Application Programming Interface). The Windows system is made up of several DLL's. You can use Paradox to access routines within the DLLs that comprise the Windows system.

Declare a **uses** block in an object's Uses window, and within that window, declare one **uses** block for each DLL you want to use. You don't have to declare every routine the DLL contains, just the ones you want to use. Once declared, routines are available to all methods attached to that object, to all objects that object contains, and to forms or libraries that reference the declarations through an ObjectPAL **uses** block.

In a **uses** block, declare the data types of parameters and return types using the following keywords:

Data type	Uses keyword	ObjectPAL type	C/C++ type type	Pascal
16-bit integer	CWORD	SmallInt	short (short int)	Integer
32-bit integer	CLONG	LongInt	long (long int)	Longint
Natural integer (*)	CLONG (*)	LongInt (*)	int	Integer
64-bit floating-point number	CDOUBLE	Number	double	Double
80-bit floating-point number	CLONGDOUBLE	Number	long double	
			Extended	
String pointer	CPTR	String	char *	Pchar
Binary or graphic data	CHANDLE	Binary, Graphic	HANDLE (Windows)	Thandle

The size of a natural integer is dependent upon the compiler you use to create your DLL. With Windows 95 and Windows NT, natural integers in C and Pascal are 32-bit integers, and map into CLONG. If your compiler uses 16-bit integers and then the arguments map into CWORD, and you must declare the arguments as CWORD.

The ObjectPAL keywords CWORD, CLONG, CDOUBLE, CLONGDOUBLE, CPTR, and CHANDLE are valid only within a DLL **uses** block. Don't use them anywhere else. They are used by Corel Paradox to convert between the more complex (and powerful) ObjectPAL data types and the corresponding data types in C or Pascal.

Note

- Do not modify any passed CPTR. If you change the contents of a string passed as a CPTR, the string must not grow beyond the size it had when it was passed to your routine.

Examples

{button ,AL(`BLE_OVERVIEW;BLE_USES;',0,"Defaultoverview",)} Related Topics

DLL uses block examples

Example1	Declaring variables for use as arguments
Example2	Using existing routines
Example3	Using ObjectPAL to call a function from the Windows API

Calling external routines

Previous versions of Windows (3.1 and earlier) and Corel Paradox used the Pascal calling convention (PASCAL). Windows 95 and Windows NT use a different calling convention. Corel Paradox supports this calling convention, STDCALL, PASCAL, and the C calling convention, CDECL. Corel Paradox defaults to STDCALL.

Convention	Push order	Restore stack	Link name	Used by
PASCAL	Left first	Callee	Uppercase	Pascal
CDECL	Right first	Caller	'_' prepended	C/C++
STDCALL	Right first	Callee	No change	Windows 95, Windows NT

When you declare routines to be called from a dynamic link library (DLL), you must match the calling convention that the routines were declared with. All calls to functions in the Windows 95 Application Programming Interface (API) or the Windows NT API are case-sensitive and require the use of the STDCALL calling convention.

Calls to functions written in Pascal should be declared with the PASCAL calling convention, and calls to C functions should be declared CDECL, unless the routines were explicitly declared to use a different convention when the DLL was compiled. For example, a C routine might be declared __stdcall, in which case you would declare it STDCALL in the **uses** block.

If you do not include a link name in the declaration, the routine name will be used in the call with any changes listed in the Link name column in the table above.

When passing a value to a C procedure, the ObjectPAL variable must be declared and typed explicitly. However, AnyType is not allowed.

All C and C++ functions that you want ObjectPAL to call must be exported in the .DEF file, or tagged with _export in the function declaration.

Using C++

Calling dynamic link library (DLL) modules written in C++ requires either the use of a C linkage specification or the mangled name in the **uses** block.

To specify a C++ function with C linkage, the modules must be in one of the following forms:

```
extern "C" declaration
extern "C" { declarations }
```

For example, if a C module contains these functions:

```
char *SCopy(char*, char*);
void ClearScreen(void);
```

they must be declared in a C++ module in one of the following ways to have a C linkage.

```
extern "C" char *SCopy(char*, char*);
extern "C" void ClearScreen(void);
```

or

```
extern "C" {
char *SCopy(char*, char*);
void ClearScreen(void);
}
```

Otherwise, you can specify the mangled name of the routine to call. The mangled name can be found by using a dumping file on the .OBJ file produced by your compiler.

For example, if a Borland C++ module (named MyLib) contains the function

```
int __cdecl MyFunction(int arg)
```

then you can use this **uses** block to declare the DLL routine.

```
uses MyLib
   MyFunction(CLONG arg) CLONG  [CDECL "@MyFunction$qi"]
enduses
```

All C or C++ functions that you want to call from ObjectPAL must be exported, either by using a .DEF file or the _export modifier. See your C or C++ compiler documentation for more information on exporting functions when creating DLLs.

Example

{button ,AL(`BLE_OVERVIEW;BLE_USES;',0,"Defaultoverview",)} Related Topics

Passing by value

The following table presents the syntax you should use when passing various data types by value to a C procedure. ObjectPAL passes and returns floating-point values by value, as required by the Borland C++ compiler. Other C compilers may have different requirements. To ensure compatibility with any C compiler, pass values by pointer.

It is assumed that these ObjectPAL variables have been declared: si SmallInt, li LongInt, nu Number, st String, gr Graphic, and bi Binary

C data type	C syntax	In uses block	ObjectPAL call
long double	void __stdcall cproc(long double value)	cproc(numvar CLONGDOUBLE)	cproc(si) cproc(li) cproc(nu)
double	void __stdcall cproc(double value)	cproc(numvar CDOUBLE)	cproc(si) cproc(li) cproc(nu)
long int	void __stdcall cproc(long int value)	cproc(longvar CLONG)	cproc(si) cproc(li)
short int	void __stdcall cproc(short int value)	cproc(shortvar CWORD)	cproc(si)
int	void __stdcall cproc(int value)	cproc(longvar CWORD)	cproc(si)
(String)	void __stdcall cproc(char * value)	cproc(stringvar CPTR)	cproc(st)
(Graphic)	void __stdcall cproc(HANDLE value)	cproc(bitmapvar CHANDLE)	cproc(gr)
(Binary)	void __stdcall cproc(HANDLE value)	cproc(binaryvar CHANDLE)	cproc(bi)

{button ,AL(`BLE_OVERVIEW;BLE_USES;',0,"Defaultoverview",)} Related Topics

Passing by pointer

When ObjectPAL passes information to a C procedure that takes pointers to information, the pointer points directly to the corresponding value in the ObjectPAL object. Variables in ObjectPAL are treated as objects internally. For example, if you want an int * and you pass a LongInt, you will get a pointer that points directly to the integer value inside the LongInt object. You can then modify the value of the LongInt using the pointer in your DLL. This could, however, corrupt ObjectPAL by overwriting memory (writing past the bounds of the memory pointer). **Use caution when using pointers**.

Use pointers to

- change the information (this should be done by function return values if possible)
- Pass floating-point values to C procedures that were not compiled using the Borland C compiler. Different C compilers use different conventions for passing and returning floating-point values (double and long double). The only way to pass compiler-independent information is by pointer.

The following table presents the syntax you should use when passing various data types by pointer to a C procedure, with the assumption that these ObjectPAL variables have been declared: si SmallInt, li LongInt, nu Number, st String, gr Graphic, and bi Binary.

C data type	C syntax	In USES block	ObjectPAL call
long double *	void __stdcall cproc(long double * value)	cproc(numvar CPTR)	cproc(nu)
long int *	void __stdcall cproc(long int * value)	cproc(longvar CPTR)	cproc(li)
int *	void __stdcall cproc(int * value)	cproc(longvar CPTR)	cproc(li)
short int *	void __stdcall cproc(short int * value)	cproc(shortvar CPTR)	cproc(si)
char *	void __stdcall cproc(char * value)	cproc(strvar CPTR)	cproc(st)

{button ,AL(`BLE_OVERVIEW;BLE_USES;',0,"Defaultoverview",)} Related Topics

Returning values

The following table presents the syntax for data type value that have been returned from a C procedure. The assumption is that these ObjectPAL variables have been declared: si SmallInt, li LongInt, nu Number, and st String

C data type	C syntax	In USES block	ObjectPAL call
long double	long double __stdcall cproc(void)	cproc() CLONGDOUBLE	nu = cproc()
double	double __stdcall cproc(void)	cproc() CDOUBLE	nu = cproc()
long int	long int __stdcall cproc(void)	cproc() CLONG	li = cproc()
short int	short int __stdcall cproc(void)	cproc() CWORD	si = cproc()
char *	char * __stdcall cproc(void)	cproc() CPTR	st = cproc()

{button ,AL(`BLE_OVERVIEW;BLE_USES;',0,"Defaultoverview",)} **Related Topics**

Notes on Graphic and Binary data (CHANDLE)

Graphic and Binary data are passed via CHANDLE—a handle to Windows memory. In C use the HANDLE typedef by including it inside code like this:

```
void __stdcall cproc(HANDLE value)
{
   // declare ptr to point to Global Memory Block
   huge *ptr = (huge *) GlobalLock(value);


   // ... make use of ptr here
   // ... DO NOT use 'GlobalFree(value);'


   GlobalUnlock(value);
}
```

For a Binary variable, HANDLE is a handle to memory that holds the information in the binary BLOB. There is no header information. As with any strings you pass, you can read or modify the data, but you cannot change its size.

For a Graphic variable, HANDLE is a Windows bitmap handle that you can use as you would any other bitmap HANDLE.

{button ,AL(`BLE_OVERVIEW;BLE_USES;',0,"Defaultoverview",)} Related Topics

DLL uses block example 1

The following example references a dynamic link library (DLL) named MYSTUFF.DLL. To use a DLL routine in a method, declare variables to use as arguments and then call the routine. For example,

```
; this goes in an object's Uses window
uses myStuff ; reads routines from MYSTUFF.DLL
   doSomething(thisNum CLONG, thatNum CLONG) CDOUBLE ; declare a routine
endUses


; this modifies an object's mouseUp method
method mouseUp(var eventInfo MouseEvent)
var
   thisNum, thatNum LongInt ; declare variables to pass to the routine
   myResult Number
endVar


thisNum = 3155111
thatNum = 5535345
myResult = doSomething(thisNum, thatNum) ; call the routine, return a result
endMethod
```

In this example, notice how the variables in the method are declared as LongInt and Number, and the arguments in the **uses** block are correspondingly declared as CLONG and CDOUBLE.

DLL uses block example 2

The following example uses routines from MINMAX.DLL, written using a 32-bit Pascal compiler. The code for the dynamic link library (DLL) is as follows:

```
library MinMax;


function Min(x, y: integer): integer; stdcall; export;
begin
   if x < y then
      result := x
   else
      result := y;
end;


function Max(x, y: integer): integer; stdcall; export;
begin
   if x > y then
      result := x
   else
      result := y;
end;


exports
   Min, Max;


begin
end.
```

The following ObjectPAL code uses the routines in the DLL. The code for the Uses window appears first, followed by the code that modifies a button's **pushButton** method:

```
; the following goes in a button's Uses window
uses
   MinMax ; load routines from MINMAX.DLL
   Min (x CLONG, y CLONG) CLONG  [STDCALL]
   Max (x CLONG, y CLONG) CLONG  [STDCALL]
endUses
```

The following code modifies a button's built-in **pushButton** method:

```
method pushButton(var eventInfo Event)
var
   x, y, z  LongInt
endVar
   x = 2
   y = 6
   z = Min(x, y)          ; call Min from the DLL
   msgInfo("Min", z)
   z = Max(x, y)          ; call Max from the DLL
   msgInfo("Max", z)
endMethod
```

DLL uses block example 3

The following example shows how to use ObjectPAL to call a function from the Windows application programming interface (API). It calls the Windows API function MessageBox to display a dialog box.

The following code is attached to a button's Uses window:

```
Uses USER32    ; The MessageBox function is in
               ; the Windows system DLL USER32.DLL
               ; usually found in C:\WINDOWS\SYSTEM
   MessageBox(hWnd CHANDLE, lpText CPTR, lpCaption CPTR, wType CLONG) CLONG
endUses
```

The following code is attached to a button's built-in **pushButton** method. It calls MessageBox, passing it zero for the window handle ensuring that it’s not connected to any particular window. The code also passes text for the message and the caption and another zero to signify an OK-style message box. The return value is ignored.

```
method pushButton(var eventInfo event)
   MessageBox(0,
              "Your message here",
              "Your caption here",
              0)
endMethod
```

For more information on the parameters for this and other Windows API function calls, see the Windows API reference .

var keyword

Declares variables.

Syntax

```
var
   [ varName [ , varName ] * varType ]*
endVar
```

Description

The **var...endVar** block declares variables by associating a variable name *varName* with a data type *varType*. When you declare more than one variable of the same type on the same line, use commas to separate the names.

A variable's scope depends on the block in which it is declared.

 **Note**

- You declare variables in a **var...endVar** block in ObjectPAL code or in the Var window on the Methods page of the Object Explorer.

Example

{button ,AL(`BLE_OVERVIEW;BLE_DECLARATION;',0,"Defaultoverview",)} Related Topics

var example

The following example demonstrates how the var keyword declares a variable.

```
var
   myChars, xx String
   myNum Number
   orders, sales, parts TCursor
   proteus AnyType
   myBox UIObject
   a, b Array[5] SmallInt
   myOtherNum Number
endVar
```

while keyword

Repeats a sequence of statements as long as a specified condition is True.

Syntax

```
while Condition
   [ Statements ]
endWhile
```

Description

while evaluates the logical expression *Condition*. If *Condition* is False, the *Statements* are not executed. If the *Condition* is True, the *Statements* between *Condition* and **endWhile** are executed in sequence. Control then returns to the top of the loop, and the *Condition* is evaluated again. The steps are repeated until the *Condition* is False, at which point the loop is exited and control advances to the next statement after **endWhile**.

You can use **loop** within the body of the **while** variable to force control back to the top of the **loop**, skipping the statements between **loop** and **endWhile**. You can also use **quitLoop** to exit the loop or nest **while** statements to any level.

while and **for** are used for different reasons. Use **for** to execute a sequence of statements a known number of times. Use **while** to execute a sequence of statements an arbitrary number of times.

Example

{button ,AL(`BLE_OVERVIEW;BLE_CONTROL_STRUCTURE;',0,"Defaultoverview",)} Related Topics

while example

The following example creates an array of last names.

```
var
   myNames TCursor
   namesArray Array[] String
   n SmallInt
endVar

myNames.open("names.db")
namesArray.grow(1)
namesArray[1] = myNames."Last name"
n=1

while myNames.nextRecord()
   n = n + 1
   namesArray.grow(1)
   namesArray[n] = myNames."Last name"
endWhile
```

Keywords

The keywords in this list cannot be used to name objects, variables, arrays, methods, or procedures. The case of the words is irrelevant; they cannot be used in any combination of uppercase or lowercase.

Generally, you should not use object type names, names of basic language elements, names of methods and procedures in the run-time library, or names of built-in event methods.

Keywords

active	endTry	proc
and	endType	query
array	endUses	quitLoop
as	endVar	record
case	endWhile	refIntStruct
caseInsensitive	for	retry
const	forEach	return
container	from	scan
create	if	secStruct
Database	iif	self
descending	in	sort
disableDefault	index	step
doDefault	indexStruct	struct
DynArray	is	subject
else	key	switch
enableDefault	lastMouseClicked	switchMenu
endConst	lastMouseRightClicked	tag
endCreate	like	then
endFor	loop	to
endForEach	maintained	try
endIf	method	type
endIndex	not	unique
endMethod	ObjectPAL	uses
endProc	of	var
endQuery	on	where
endRecord	onFail	while
endScan	or	with
endSort	otherwise	without
endSwitch	passEvent	
endSwitchMenu	primary	

{button ,AL(`BLE_OVERVIEW;BLE_OVERVIEW;OPAL_METH_SYENUMRTLCLASSNAMES;OPAL_METH_SYENUMRTLCONSTANTS;OPAL_METH_SYENUMRTLMETHODS;',0,"Defaultoverview",)} Related Topics

Built-in object variables

ObjectPAL provides built-in object variables that you can use to refer to UIObjects. These variables are particularly useful for creating generalized code. For example, when the following statement executes, it sets the color of the active object (the object that has focus). You don't have to specify the object by name.

```
active.Color = Blue
```

The built-in object variables are

active
container
lastMouseClicked
lastMouseRightClicked
self
subject

{button ,AL(`BLE_OVERVIEW;BLE_OVERVIEW;',0,"Defaultoverview",)} **Related Topics**

Properties and property values

Click one of the following properties for more information:

Alignment
Arrived
AttachedHeader
AutoAppend
AvgCharSize
BlankRecord
Border
BottomBorder (read-only)
Breakable
ButtonType
ByRows
CalculatedField
Caption
CenterLabel
CheckedValue
Class
Color
Columnar
ColumnPosition
ColumnWidth
CompleteDisplay
ContainerName
ControlMenu
CurrentColumn
CurrentPage
CurrentRecordMarker.Color
CurrentRecordMarker.LineStyle
CurrentRecordMarker.Show
CurrentRow
CursorColumn
CursorLine
CursorPos
DataSource
Default
DefineGroup
DeleteColumn
Deleted
DeleteWhenEmpty
Design.ContainObjects
Design.PinHorizontal
Design.PinVertical
Design.Selectable
KeyField
LabelText
LeftBorder (read-only)
Line.Color
Line.LineStyle
Line.Thickness
LineEnds
LineSpacing
LineSqueeze
LineStyle
LineType
List.Count
List.Selection
List.Value
Locked
LookupTable
LookupType
Magnification
Manager
Margins.Bottom
Margins.Left
Margins.Right
Margins.Top
MarkerPos
MaximizeButton
Maximum
MemoView
MinimizeButton
Minimum
Modal
MouseActivate
Name
NCols
Next
NextTabStop
NoEcho
NRecords
NRows
NumberPages
Orphan/
Windowopal_prop_orphanwidow
OtherBandName
OverStrike

Design.SizeToFit
DesignModified
DesignSizing
DesktopForm
DialogForm
DialogFrame
DisplayType
DrillDown
Editing
Enabled
End
FieldName
FieldNo
FieldRights
FieldSize
FieldSqueeze
FieldType
FieldUnits2
FieldValid
FieldView
First
FirstRow
FitHeight
FitWidth
FlatLook
FlyAway
Focus
Font.Color
Font.Scriptopal_prop_font.script
Font.Size
Font.Style
Font.Typeface
Format.DateFormat
Format.LogicalFormat
Format.NumberFormat
Format.TimeFormat
Format.TimeStampFormat
Frame.Color
Frame.Style
Frame.Thickness
FrameObjects
FullName
FullSize
Grid.Color
Grid.GridStyle
Grid.RecordDivider
GridLines.Color
GridLines.ColumnLines
GridLines.HeadingLines
Owner
PageSize
PageTiling
Pattern.Color
Pattern.Style
PersistView
Picture (enhanced for 5.0)
PinHorizontal
PinVertical
PositionalOrder
Position
PrecedePageHeader
Prev
PrinterDocument
PrintOn1stPage
ProgID
Range
RasterOperation
Readonly
RecNo
Refresh
RefreshOption
RemoveGroupRepeats
RepeatHeader
Required
RightBorder (read-only)
RowHeight
RowNo
Scroll
SeeMouseMove
SelectedText
Select
SeqNo
ShowAllColumns
ShowGrid
Shrinkable
Size
SizeToFit
SnapToGrid
SortOrder
SpecialField
SquareTabs
StandardMenu
StandardToolbar
Start
StartPageNumbers
Style
SummaryModifier
TabHeight

GridLines.LineStyle
GridLines.QueryLook
GridLines.RowLines
GridLines.Spacing
GridValue
GroupObjects
GroupRecords
Header
HeadingHeight
Headings
HorizontalScrollBar
HTMLAction
HTMLFormParams
HTMLMethod
InactiveColor
IncludeAllData
IndexField
InsertColumn
InsertField
Inserting
Invisible
Justification
TableName
TabsAcross
TabsOnTop
TabStop
Text
ThickFrame
Thickness
Title
TopBorder (read-only)
TopLine
Touched
Translucent
UncheckedValue
Value
VerticalScrollBar
Visible
WideScrollBar
Width
WordWrap
Xseparation
Yseparation

{button ,AL(`OBJECT_PROPERTIES;',0,"Defaultoverview",)} **Related Topics**

Alignment property

Data type

SmallInt

Description

Specifies the position of text relative to a field or text object.

Values

TextAlignCenter, TextAlignJustify, TextAlignLeft, TextAlignRight

Arrived property

Data type

Logical

Description

Specifies whether the focus has arrived at the object.

Values

True, False

AttachedHeader property

Data type

Logical

Description

Determines whether a table frame's header is attached to the table frame. If the AttachedHeader property is set to True, the header is attached. If this property is set to False, the header is not attached and selection handles surround the header object.

Values

True, False

AutoAppend property

Data type

Logical

Description

Determines whether Corel Paradox insets a blank record when you move the cursor past then end of the table during editing. If AutoAppend is set to True (default), Corel Paradox inserts a blank record. If AutoAppend is set to False, Corel Paradox does not insert a blank record.

Setting an object's AutoAppend property has the same effect as right-clicking a table in the Data Model dialog box and choosing Auto-Append from its menu. This property is valid for an object that is bound to a table in the data model.

Values

True, False

AvgCharSize property

Data type

Point

Description

Specifies the average width and height of a character in the current font. AvgCharSize is a read-only property.

Values

>0

BlankRecord property

Data type

Logical

Description

Reports whether a record is blank.

Values

True, False

Border property

Data type

Logical

Description

Reports whether a form's window has a border.

Values

True, False

BottomBorder property

Data type

LongInt

Description

Returns the size of an object's bottom border (in twips). You cannot embed another object within an object's border.

Values

N/A

Breakable property

Data type

Logical

Description

Specifies whether an object can be divided across page breaks in a report. The breakable property is read-only for elliptical lines.

Values

True, False

ButtonType property

Data type

SmallInt

Description

Specifies a button's display type.

Values

CheckBoxType, PushButtonType, RadioButtonType

ByRows property

Data type

Logical

Description

Determines the layout of a multi-record object in a form or report. If the ByRows property is set to True, record layout is top-down, then left-right. If the ByRows property is set to False, record layout is left-right, then top-down.

Values

True, False

CalculatedField property

Data type

Logical

Description

Specifies whether a field is a calculated field.

Values

True, False

Caption property

Data type

Logical

Description

Reports whether a form has a caption or Title Bar.

Values

True, False

CenterLabel property

Data type

Logical

Description

Specifies whether a label is centered on a button.

Values

True, False

CheckedValue property

Data type

String

Description

Specifies the string that the check box or radio button writes to its parent field object when it is chosen.

Values

N/A

Class property

Data type

String

Description

Returns a UIObject's class.

Values

Band, Bitmap, Box, Button, Cell, Chart, Crosstab, EditRegion, Ellipse, Field, Form, FormData, Group, Header, Line, List, Multi-record, OLE, Page, Record, Report, ReportPrint, TableFrame, Text

Color property

Data type

LongInt

Description

Specifies an object's display color.

Values

Black, Blue, Brown, DarkBlue, DarkCyan, DarkGray, DarkGreen, DarkMagenta, DarkRed, Gray, Green, LightBlue, Magenta, Red, White, Yellow, Transparent

Columnar property

Data type

Logical

Description

Determines whether a multi-record object displays it records in a fixed grid.

When the Columnar property is set to True, each record expands or contracts when you print or preview the report. This means that the multi-record object does not display the records in a fixed-size grid. When the Columnar property is set to False, the multi-record object displays its records in a fixed grid. You can fit more records on a single page when the Columnar property is set to False.

The Columnar property is valid for multi-record objects in Report Design windows. It is not valid for objects in forms.

Values

True, False

ColumnPosition property

Data type

SmallInt

Description

Specifies a new position (starting with 1) for the active column in a table frame.

Values

>0

ColumnWidth property

Data type

LongInt

Description

Specifies the width (in twips) of the active column in a table frame.

Values

>0

CompleteDisplay property

Data type

Logical

Description

Specifies whether to display the complete contents in a field.

Values

True, False

ContainerName property

Data type

String

Description

Reports the name of an object's container.

Values

N/A

ControlMenu property

Data type

Logical

Description

Specifies whether a form has a Control menu. If the ControlMenu property is set to True (default), the form has a control menu; otherwise, it does not. ControlMenu is a read/write property.

Values

True, False

CurrentColumn property

Data type

SmallInt

Description

Determines the current column in crosstabs, multi-record objects, and table frames at run time. For example, because all cells in a crosstab have the same name, it is impossible to address a single cell. You can use CurrentColumn and CurrentRow to move to a particular cell you and have your code address the active object, its container, and so on.

CurrentColumn also allows you to control column rotation and sizing in table frames at design time. Setting this property in the designer does not necessarily affect the visual appearance. Table frames do not have to be selected.

Values

>0

CurrentPage property

Data type

String

Description

Returns the active page of the form or notebook, including container objects.

CurrentRecordMarker.Color property

Data type

LongInt

Description

Specifies the display color of the active record in a table view.

Values

Black, Blue, Brown, DarkBlue, DarkCyan, DarkGray, DarkGreen, DarkMagenta, DarkRed, Gray, Green, LightBlue, Magenta, Red, White, Yellow, Transparent

CurrentRecordMarker.LineStyle property

Data type

SmallInt

Description

Specifies the style of the line that marks the active record in a table view.

Values

DashDotDotLine, DashDotLine, DashedLine, DottedLine, NoLine, SolidLine

CurrentRecordMarker.Show property

Data type

Logical

Description

Specifies whether to highlight the active record in a table view.

Values

True, False

CurrentRow property

Data type

SmallInt

Description

Determines the active row in crosstabs, multi-record objects, and table frames at run time. For example, because all cells in a crosstab have the same name, it is impossible to address a single cell. You can use CurrentColumn and CurrentRow to move to a particular cell you and have your code address the active object, its container, and so on.

CurrentRow also allows you to control row rotation and sizing in table frames at design time. Setting this property in the designer does not necessarily affect the visual appearance. Table frames do not have to be selected.

Values

>0

CursorColumn property

Data type

LongInt

Description

Specifies the horizontal position of the cursor in a field object, where position 0 lies to the left of the first character.

Values

N/A

CursorLine property

Data type

LongInt

Description

Specifies the vertical position of the cursor in a field object, where the first line is line 1.

Values

N/A

CursorPos property

Data type

LongInt

Description

Specifies the position of the insertion point in a field object, relative to the first character in the field. Counting begins with 0, the position to the left of the first character.

Values

N/A

DataSource property

Data type

String

Description

Specifies the name of the table that supplies the items in a list. DataSource fills a list with values from a specified field or column in a table.

Values

N/A

Default property

Data type

String

Description

Specifies a field's default value.

Values

N/A

DefineGroup property

Data type

Logical

Description

Specifies whether a report band defines a group.

Values

True, False

DeleteColumn property

Data type

SmallInt

Description

Specifies the column to delete from a table frame. DeleteColumn is a write-only property.

Values

>0

Deleted property

Data type

Logical

Description

Reports whether a record in a dBASE table has been deleted.

Values

True, False

DeleteWhenEmpty property

Data type

Logical

Description

Determines whether a field is empty and can be deleted from the report. If the DeleteWhenEmpty property is set to True, the field, including all labels, buttons, and attributes, is deleted.

Values

True, False

Design.ContainObjects property

Data type

Logical

Description

Specifies whether an object can contain other objects.

Values

True, False

Design.PinHorizontal property

Data type

Logical

Description

Specifies whether to prevent an object from moving horizontally.

Values

True, False

Design.PinVertical property

Data type

Logical

Description

Specifies whether to prevent an object from moving vertically.

Values

True, False

Design.Selectable property

Data type

Logical

Description

Specifies whether an object can be selected. The Design.Selectable property is valid for UIObjects in forms and reports in design windows and at runtime. When the Design.Selectable property is set to True, you can select the object, and selection handles appear around the object. When the Design.Selectable property is set to False, you cannot select the object, so there are no handles. You can still right-click the object to view its menu.

Values

True, False

Design.SizeToFit property

Data type

Logical

Description

Specifies whether the object resizes to accommodate its contents.

Values

True, False

DesignModified property

Data type

Logical

Description

Specifies whether a form or report has been modified. Corel Paradox sets the DesignModified property to True when the form or report's design is changed. When you close a form or report and the DesignModified property is set to True, Corel Paradox prompts you to save your changes. When the DesignModified property is set to False, Corel Paradox closes the form or report, and all changes are lost.

Values

True, False

DesignSizing property

Data type

SmallInt

Description

Specifies design time sizing for a text box.

Values

TextFixedSize, TextGrowOnly, TextSizeToFit

DesktopForm property

Data type

Logical

Description

Specifies whether a form's menus are used by other forms on the desktop.

Values

True, False

DialogForm property

Data type

Logical

Description

Specifies whether a form opens as a dialog box.

Values

True, False

DialogFrame property

Data type

Logical

Description

Specifies whether a form has a conventional dialog box frame. When the DialogFrame property is set to True, and when DialogForm and Border are also True, the form has a conventional dialog box frame.

Values

True, False

DisplayType property

Data type

SmallInt

Description

Returns a field object's display type.

Values

CheckBoxField, ComboField, EditField, LabeledField, ListField, RadioButtonField

DrillDown property

Data type

Integer

Description

Specifies the detail included in a report that is output to an HTML template file. DrillDown is a property of a report group band and is specific to each group band. Setting this property generates a BDE_INDEX tag in the HTML template file.

Value	Description
0	Creates a single HTML template file for the entire report
1	Creates separate HTML template files for each group in the drilldown
2	Creates separate HTML template files for each nested group in the drilldown
3	Create separate HTML template files for each item in the drilldown

Values

0, 1, 2, 3

Editing property

Data type

Logical

Description

Specifies whether an object's table is in Edit mode. Editing returns True if the table associated with an object is in Edit mode. For field objects, Editing specifies whether the field is active and using a temporary edit object (e.g., field view). Editing is valid for objects including forms bound to tables.

When you put a form in Edit mode (e.g., by pressing F9 or clicking the Edit Data button on the Toolbar), all associated tables are also put in Edit mode.

Values

True, False

Enabled property

Data type

Logical

Description

Specifies whether a UI object is enabled (true) or disabled (false). When the Enabled property is set to False, the UI object is disabled and its text is grayed. The object does not responds to mouse clicks and you cannot move to it using the TAB key. All objects contained by a disabled UI object are also disabled or set to FALSE.

Values

True, False

End property

Data type

Point

Description

Specifies the coordinates of the end of a line. To specify the coordinates of the start of a line, see the Start property.

Values

N/A

FieldName property

Data type

String

Description

Specifies the name of the field to which a field object or list is bound.

In addition to setting a FieldName value like Quant or Bookord->Quant or [Bookord.Quant], you can also specify sum(Bookord.Quant), to allow aggregated values to be placed. This property's return value now a string just like you see when you click the Copy Field button in the Define Field dialog box.

To use these new features, do either of the following:

- Change code that uses expressions like `active.TableName + "." + active.FieldName` to `active.FieldName`.
- Use the following types of calculations:

```
actField = active.FieldName

x = actField.SubStr( actField.search( "." ) + 1, actField.Size() )
```

Values

N/A

FieldNo property

Data type

SmallInt

Description

Reports a field's position in a table, where the first field is field 1.

Values

N/A

FieldRights property

Data type

String

Description

Reports the user's field rights.

Values

ReadOnly, ReadWrite, All

FieldSize property

Data type

SmallInt

Description

Specifies the size of alphanumeric and dBASE number fields.

Values

N/A

FieldSqueeze property

Data type

Logical

Description

Specifies whether to push or pull an embedded object so that text will be positioned properly within it at run time. FieldSqueeze is available only for text objects on reports.

Values

True/False

FieldType property

Data type

String

Description

Specifies the a field's data type.

Values

N/A

FieldUnits2 property

Data type

SmallInt

Description

Specifies the number of decimal places in a dBASE number field. For a Corel Paradox table (and any other driver or field type that does not require field unit specifications), FielsUnits2 is 0.

Values

N/A

FieldValid property

Data type

Logical

Description

Reports whether a field passes its own value checks.

Values

True, False

FieldView property

Data type

Logical

Description

Reports whether a field is in field view.

Values

True, False

First property

Data type

String

Description

Returns the name of the first child object in a container.

Values

N/A

FirstRow property

Data type

String

Description

FirstRow returns the name of the first record object in a table frame or multi-record object. FirstRow is a read-only property.

Values

N/A

FitHeight property

Data type

Logical

Description

Specifies whether an edit region expands vertically to accommodate text.

Values

True, False

FitWidth property

Data type

Logical

Description

Specifies whether an edit region or crosstab cell expands horizontally to accommodate text.

Values

True, False

FlatLook property

Data type

Logical

Description

Specifies whether a field object or button appears to be 3-Dimensional, or flat.

Values

True, False

FlyAway property

Data type

Logical

Description

Reports whether a record has moved to its sorted position in a table.

Values

True, False

Focus property

Data type

Logical

Description

Reports whether an object's built-in **setFocus** method has been called. The Focus property is set to False when the object's built-in **removeFocus** method is called.

Values

True, False

Font.Color property

Data type

LongInt

Description

Specifies the color of characters displayed in a field or text object.

Values

Black, Blue, Brown, DarkBlue, DarkCyan, DarkGray, DarkGreen, DarkMagenta, DarkRed, Gray, Green, LightBlue, Magenta, Red, White, Yellow, Transparent

Font.Script property

Data type

String

Description

The language script or character set of the selected font.

Values

Varies according to individual system fonts.

Font.Size property

Data type

SmallInt

Description

Specifies (in printer's points) the size of characters in a field or text object.

Values

>0

Font.Style property

Data type

SmallInt

Description

Specifies the style of characters displayed in a field or text object.

Values

FontAttribBold, FontAttribItalic, FontAttribNormal, FontAttribStrikeout, FontAttribUnderline

Font.Typeface property

Data type

String

Description

Specifies the typeface of characters displayed in a field or text object.

Values

Varies according to individual system attributes.

Format.DateFormat property

Data type
String

Description
Specifies the format for date values.

Values
Format specification

Format.LogicalFormat property

Data type

String

Description

Specifies the format for logical values.

Values

Format specification

Format.NumberFormat property

Data type

String

Description

Specifies the format for number values.

Values

Format specification

Format.TimeFormat property

Data type

N/A

Description

Specifies the format for time values.

Values

Format specification

Format.TimeStampFormat property

Data type

String

Description

Specifies the format for time stamps.

Values

Format specification

Frame.Color property

Data type

LongInt

Description

Specifies the color of an object's frame.

Values

Black, Blue, Brown, DarkBlue, DarkCyan, DarkGray, DarkGreen, DarkMagenta, DarkRed, Gray, Green, LightBlue, Magenta, Red, White, Yellow, Transparent

Frame.Style property

Data type

SmallInt

Description

Specifies the style of an object's frame.

Values

3DWindows, 3DWindowsGroup, DashDotDotFrame, DashDotFrame, DashedFrame, DottedFrame, DoubleFrame, Inside3DFrame, NoFrame, Outside3DFrame, ShadowFrame, SolidFrame, WideInsideDoubleFrame, WideOutsideDoubleFrame

Frame.Thickness property

Data type

SmallInt

Description

Specifies the thickness of an object's frame in pixels.

Values

N/A

FrameObjects property

Data type

Logical

Description

Specifies whether a dotted frame is displayed around objects in the designers. If the FrameObjects property is set to, it creates a 1 pixel region in which embedding cannot occur.

Values

True, False

FullName property

Data type

String

Description

Returns the full name, including containership path, of form object.

Values

N/A

FullSize property

Data type

Point

Description

Specifies the full size of a scrolling object.

Values

N/A

Grid.Color property

Data type

LongInt

Description

Specifies the color of the grid in a table frame.

Values

Black, Blue, Brown, DarkBlue, DarkCyan, DarkGray, DarkGreen, DarkMagenta, DarkRed, Gray, Green, LightBlue, Magenta, Red, White, Yellow, Transparent

Grid.GridStyle property

Data type

SmallInt

Description

Specifies the style of the gridlines in a table frame.

Values

tf3D, tfDoubleLine, tfNoGrid, tfSingleLine, tfTripleLine

Grid.RecordDivider property

Data type

Logical

Description

Specifies whether dividing lines are displayed between records in a table frame.

Values

True, False

GridLines.Color property

Data type

LongInt

Description

Specifies the color of gridlines in a Table window.

Values

Black, Blue, Brown, DarkBlue, DarkCyan, DarkGray, DarkGreen, DarkMagenta, DarkRed, Gray, Green, LightBlue, Magenta, Red, White, Yellow, Transparent

GridLines.ColumnLines property

Data type

Logical

Description

Specifies whether column lines are displayed in a Table window.

Values

True, False

GridLines.HeadingLines property

Data type

Logical

Description

Specifies whether heading lines are displayed in a Table window.

Values

True, False

GridLines.LineStyle property

Data type

SmallInt

Description

Specifies the style of gridlines in a Table window.

Values

DashDotDotLine, DashDotLine, DashedLine, DottedLine, NoLine, SolidLine

GridLines.QueryLook property

Data type

Logical

Description

Specifies whether a Table window displays gridlines in the same style as a Query Editor window.

Values

True, False

GridLines.RowLines property

Data type

Logical

Description

Specifies whether to display gridlines in a Table window.

Values

True, False

GridLines.Spacing property

Data type

SmallInt

Description

Specifies the spacing between gridlines in a Table window.

Values

TextSingleSpacing, TextDoubleSpacing, TextTripleSpacing

GridValue property

Data type

Point

Description

Determines the minimum grid interval setting in twips. Only the UI can control the number of grid intervals, displayed as major or minor, when the ShowGrid option is enabled. However, GridValue gives more control of the grid's granularity.

Values

>0

GroupObjects property

Data type

Logical

Description

Specifies whether to group selected objects in forms and reports. You can also group and ungroup selected objects in forms and reports using the Group and Ungroup menu items. GroupObjects is a write-only property.

Values

True, False

GroupRecords property

Data type

SmallInt

Description

Determines the number of records grouped together in a report.

Values

>0

Header property

Data type

String

Description

Returns the name of a table frame's header object (if it has one). Header is a read-only property

Values

N/A

HeadingHeight property

Data type

LongInt

Description

Determines the height (in twips) of the heading in a Table window.

Values

>0

Headings property

Data type

String

Description

Specifies which report headings to print.

Values

GroupOnly, PageAndGroup

HTMLAction property

Data type

String

Description

Specifies the value of a <Form> tag's ACTION parameter in the HTML template file. HTMLAction defaults to the form object’s noise name (e.g., #Form1), and is used for HTML forms only.

Values

N/A

HTMLFormParams property

Data type

Logical

Description

Specifies whether a form's fields are INPUT fields (with the BDE_PARAM tag), or readOnly fields. If any fields on the form are input fields, this property is set to True. You can set individual fields to readOnly as needed.

Values

True, False

HTMLMethod property

Data type

String

Description

Specifies the value of the <Form> tag's METHOD parameter in the HTML template file. This property defaults to POST and is used for HTML forms only.

Values

POST, GET

HorizontalScrollBar property

Data type

Logical

Description

Specifies whether a table frame has a horizontal scroll bar.

Values

True, False

InactiveColor property

Data type

LongInt

Description

Specifies the color of the notebook page when it is not selected.

Values

Black, Blue, Brown, DarkBlue, DarkCyan, DarkGray, DarkGreen, DarkMagenta, DarkRed, Gray, Green, LightBlue, Magenta, Red, White, Yellow, any user-defined color.

IncludeAllData property

Data type

Logical

Description

Specifies whether all data in a table is included.

Values

True, False

IndexField property

Data type

Logical

Description

Reports whether a field object is bound to an indexed field in a table.

Values

True, False

InsertColumn property

Data type

SmallInt

Description

Specifies where to insert a column in a table frame. InsertColumn is a write-only property.

Values

>0

InsertField property

Data type

Point

Description

Inserts a field object into a text box. The field's size is specified using a Point value, and its upper-left corner is positioned at (0, 0) (relative to the text box). InsertField is a write-only property.

Values

N/A

Inserting property

Data type

Logical

Description

Returns True when a record is inserted anywhere in a form.

Values

True, False

Invisible property

Data type

Logical

Description

Determines whether an object is visible at run time. Invisible applies to boxes and lines in reports. Unlike Visible, when you use Invisible to hide an object, contained objects are not hidden. Set Invisible to True to hide an object; otherwise, objects are visible.

Values

True, False

Justification property

Data type

SmallInt

Description

Specifies the justification of data in a Table window.

Values

TextAlignTop, TextAlignBottom, TextAlignVCenter, TextAlignLeft, TextAlignRight, TextAlignCenter

KeyField property

Data type

Logical

Description

Reports whether a field object is bound to a key field in a table.

Values

True, False

LabelText property

Data type

String

Description

Specifies the text that is displayed on a button label.

Values

N/A

LeftBorder property

Data type

LongInt

Description

Returns the size of an object's left border (in twips). You cannot embed another object within an object's border.

Values

N/A

Line.Color property

Data type

LongInt

Description

Specifies the line color of an ellipse.

Values

Black, Blue, Brown, DarkBlue, DarkCyan, DarkGray, DarkGreen, DarkMagenta, DarkRed, Gray, Green, LightBlue, Magenta, Red, White, Yellow, Transparent

Line.LineStyle property

Data type

SmallInt

Description

Specifies the line style of an ellipse.

Values

DashDotDotLine, DashDotLine, DashedLine, DottedLine, NoLine, SolidLine

Line.Thickness property

Data type

SmallInt

Description

Specifies the line thickness of an ellipse.

Values

N/A

LineEnds property

Data type

SmallInt

Description

Specifies whether to place arrows at the ends of a line.

Values

ArrowBothEnds, ArrowOneEnd, NoArrowEnd

LineSpacing property

Data type

SmallInt

Description

Specifies the number of blank lines to print between each line of text in a field or text object.

Values

TextDoubleSpacing, TextDoubleSpacing2, TextSingleSpacing, TextSingleSpacing2, TextTripleSpacing

LineSqueeze property

Data type

Logical

Description

Specifies whether a line that contains an empty field is blanked out at run time. LineSqueeze is available only for text objects on reports, when only one field is embedded in the text object.

Values

True/False

LineStyle property

Data type

SmallInt

Description

Specifies a line's style.

Values

DashDotDotLine, DashDotLine, DashedLine, DottedLine, NoLine, SolidLine

LineType property

Data type
SmallInt

Description
Specifies a line's type.

Values
CurvedLine, StraightLine

List.Count property

Data type

SmallInt

Description

Specifies the number of items in a list.

Values

N/A

List.Selection property

Data type

SmallInt

Description

Specifies the item selected from a list.

Values

N/A

List.Value property

Data type

AnyType

Description

Determines an item's value in a list.

Values

N/A

Locked property

Data type

Logical

Description

Reports whether the table that is bound to a design object is locked.

Values

True, False

LookupTable property

Data type

String

Description

Specifies the name of the lookup table for a field object.

Values

N/A

LookupType property

Data type
String

Description
Specifies the type of table lookup.

Values
JustCurrentField, AllCorresponding

Magnification property

Data type

SmallInt

Description

Specifies a bitmap object's display magnification. You can also specify display magnification using literal values.

Values

Magnify25, Magnify50, Magnify100, Magnify200, Magnify400, MagnifyBestFit

Manager property

Data type
String

Description
Returns a form's UIObject name.

Values
N/A

MarkerPos property

Data type

LongInt

Description

Specifies the other end of a selection. For more information, see CursorPos.

Values

N/A

Margins.Bottom property

Data type

LongInt

Description

Determines the height of a report's bottom margin in twips.

Values

>0

Margins.Left property

Data type

LongInt

Description

Determines the width of a report's left margin in twips.

Values

>0

Margins.Right property

Data type

LongInt

Description

Determines the width of a report's right margin in twips.

Values

>0

Margins.Top property

Data type

LongInt

Description

Determines the height of a report's top margin in twips.

Values

>0

MaximizeButton property

Data type

Logical

Description

Determines whether a form's window has a maximize box.

Values

True, False

Maximum property

Data type

String

Description

Specifies the maximum value allowed in a field.

Values

N/A

MemoView property

Data type

Logical

Description

Specifies whether a field object is in Memo View mode.

Values

True, False

MinimizeButton property

Data type

Logical

Description

Specifies whether a form's window has a minimize box.

Values

True, False

Minimum property

Data type

String

Description

Specifies the minimum value allowed in a field.

Values

N/A

Modal property

Data type

Logical

Description

Specifies whether a dialog box is modal. A modal dialog box retains focus until you close it, and cannot be resized or moved.

Values

True, False

MouseActivate property

Data type

Logical

Description

Specifies whether a dialog box gets focus because of a MouseEvent.

Values

True, False

NCols property

Data type

SmallInt

Description

Returns the number of columns in a table frame or multi-record object.

Values

N/A

NRecords property

Data type

LongInt

Description

Reports the number of records in the table bound to a design object. This property returns the number of records in the underlying table only. It does not return the number of records in the object.

Note

- To retrieve the number of records in a filtered set, attach a TCursor to the UIObject and call **cCount**. When you read NRecords after setting a range, the returned value represents the number of records in the set defined by the range.

Values

N/A

NRows property

Data type

SmallInt

Description

Returns the number of rows in a table frame or multi-record object.

Values

N/A

Name property

Data type

String

Description

Specifies a design object's name.

Values

N/A

Next property

Data type

String

Description

Returns the name of the next object in the same container.

Values

N/A

NextTabStop property

Data type

String

Description

Determines the object name of the next tab stop on a form or report. When you read NextTabStop, it returns a UIObject. When you write NextTabStop, you must assign a String.

Values

N/A

NoEcho property

Data type

Logical

Description

Specifies whether characters in a field object are displayed.

Values

True, False

NumberPages property

Data type

SmallInt

Description

Returns the number of pages in a notebook.

Orphan/Widow property

Data type

Logical

Description

Specifies whether a report protects against widows and orphans in breakable text objects. A widow or orphan is a single line of text that is displayed on a different page than the main body of text.

Values

True/False

OtherBandName property

Data type

String

Description

Returns the name of a report band's counterpart (the other band in a pair of bands). Given a header band OtherBandName returns the name of the footer band. Given a footer band, OtherBandName returns the name of the header band. Given a record band, OtherBandName returns the name of that record band.

Values

N/A

OverStrike property

Data type
Logical

Description
Specifies whether a field or text object is in overstrike mode. Overstrike mode is the opposite of insert mode.

Values
True, False

Owner property

Data type

String

Description

Specifies the name of an object's logical container.

Values

N/A

PageSize property

Data type

Point

Description

Determines the size of a report's page in a design window. This property exists because there is no page object in a banded report designer.

Values

>0

PageTiling property

Data type

SmallInt

Description

Determines how to arrange pages in a form. This property uses PageTiling constants.

Values

StackPages, TileHorizontal, TileVertical

Pattern.Color property

Data type

LongInt

Description

Specifies a pattern's color.

Values

Black, Blue, Brown, DarkBlue, DarkCyan, DarkGray, DarkGreen, DarkMagenta, DarkRed, Gray, Green, LightBlue, Magenta, Red, White, Yellow, Transparent

Pattern.Style property

Data type

SmallInt

Description

Specifies a pattern's style.

Values

BricksPattern, CrosshatchPattern, DiagonalCrosshatchPattern, DottedLinePattern, EmptyPattern, FuzzyStripesDownPattern, HeavyDotPattern, HorizontalLinesPattern, LatticePattern, LeftDiagonalLinesPattern, LightDotPattern, MaximumDotPattern, MediumDotPattern, RightDiagonalLinesPattern, ScalesPattern, StaggeredDashesPattern, ThickHorizontalLinesPattern, ThickStripesDownPattern, ThickStripesUpPattern, ThickVerticalLinesPattern, VerticalLinesPattern, VeryHeavyDotPattern, WeavePattern, ZigZagPattern

PersistView property

Data type

Logical

Description

Specifies whether to remain in Field or Memo view.

Values

True, False

Picture property

Data type

String

Description

Specifies a template that formats a field's value. You can use picture string characters to define a picture for a bound or unbound field object in a form. The maximum length of an ObjectPAL picture string is 255 characters. The maximum size of a picture defined interactively is 175 characters. If a bound field object has a picture defined in the data model, the Picture property does not override it.

Values

N/A

PinHorizontal property

Data type

Logical

Description

Specifies whether the crosstab object can move horizontally at run time. This property is a run-time property; the equivalent design property is named Design.PinHorizontal.

Values

True, False

PinVertical property

Data type

Logical

Description

Specifies whether the crosstab object can move vertically at run time. This property is a run-time property; the equivalent design property is named Design.PinVertical.

Values

True, False

Position property

Data type

Point

Description

Specifies the coordinates of the upper-left corner of a design object, relative to its container.

Values

N/A

PositionalOrder property

Data type

SmallInt

Description

Specifies the page number for Page objects and specifies the band position (counting from the top) for Band objects. The Report header has a PositionalOrder of 1, the Page header has a PositionalOrder of 2, and so on. Only the PositionalOrder of group bands can be set. This property can be used to rearrange groups or to swap headers and footers.

Values

>0

PrecedePageHeader property

Data type

Logical

Description

Specifies whether a report band is displayed before the page header.

Values

True, False

Prev property

Data type

String

Description

Returns the name of the previous object in the same container.

Values

N/A

PrinterDocument property

Data type

Logical

Description

Specifies whether the document is designed for printer or screen. This property setting determines the fonts and frame widths that are available for the various frame styles.

Values

True, False

PrintOn1stPage property

Data type

Logical

Description

Specifies whether to print a report band on the first page of the report.

Values

True, False

ProgID property

Data type

String

Description

Specifies the internal identifier of a system object or OLE Automation object. When the object is a Corel Paradox native Windows control (NWC) (e.g., the TrackBar or ProgressBar), the ProgID returns the string literally. When the object is an ActiveX, ProgID returns the OLE 2.0 ProgID.

Values

N/A

Range property

Data type

SmallInt

Description

Convert a group on a field to a group on a range of that field. Range is also used to change the range on a range group. Semantics are the same as the range value in the Define Group dialog box (dependent on field type). The DateRangeType constants are provided for date and timestamp fields: ByDay, ByWeek, ByMonth, ByQuarter, and ByYear.

Values

Varies according to field type.

RasterOperation property

Data type

LongInt

Description

Specifies how to blend colors in two overlapping design objects.

Values

MergePaint, NotSourceCopy, NotSourceErase, SourceAnd, SourceCopy, SourceErase, SourceInvert, SourcePaint

Readonly property

Data type

Logical

Description

Specifies whether a field object is read-only.

Values

True, False

RecNo property

Data type

LongInt

Description

Reports a record's position. Defining the record position in dBASE tables can be time consuming.

Values

N/A

Refresh property

Data type

Logical

Description

Reports when data displayed on screen changes across a network, by an ObjectPAL statement, or by a user action.

Values

True, False

RefreshOption property

Data type

SmallInt

Description

Determines what to do when data changes while printing a report. This property uses ReportPrintRestart constants.

Values

PrintFromCopy, PrintLock, PrintNoLock, PrintRestart, PrintReturn

RemoveGroupRepeats property

Data type

Logical

Description

Retains or suppresses repeated group values within a record band. When the RemoveGroupRepeats property is set to False, Corel Paradox displays of the grouped field value for each record (including duplicates) in the record band. When the RemoveGroupRepeats property is set to True, Corel Paradox prints the value of the group's first record only.

Values

True, False

RepeatHeader property

Data type

Logical

Description

Determines whether the header is repeated on each page in a report. If the RepeatHeader property is set to True, the header is repeated; otherwise, the header is not repeated.

Values

True, False

Required property

Data type

Logical

Description

Reports whether a field object have an assigned value for the record to be valid.

Values

True, False

RightBorder property

Data type

LongInt

Description

Returns the size of an object's right border (in twips). You cannot embed another object within an object's border.

Values

N/A

RowHeight property

Data type

LongInt

Description

Determines the height (in twips) of a row in a Table window.

Values

>0

RowNo property

Data type

SmallInt

Description

Reports the row number of a record displayed in a table frame, multi-record object, or table view. Counting starts at 1.

Values

N/A

Scroll property

Data type

Point

Description

Specifies the distance you've scrolled.

Values

N/A

SeeMouseMove property

Data type

Logical

Description

Specifies whether a form responds to mouse movements when it does not have focus. When the SeeMouseMove property is set to True, the form responds to mouse movements (mouseEnter, mouseMove, and mouseExit) even if it does not have focus. When the SeeMouseMove property is set to False (default), the form does not respond to mouse movements unless it has focus. This property is saved with the form.

Values

True, False

Select property

Data type

Logical

Description

Specifies whether an object is selected. Select is valid for UIObjects in a Form Design or Report Design window, but is not valid while the form or report is running.

Values

True, False

SelectedText property

Data type

String

Description

Returns the selected text in a field object.

Values

N/A

SeqNo property

Data type

LongInt

Description

Specifies the sequence number of a record, taking filters and indexes into account. This property returns <N/A> when the table is from a dynaset or has a filter.

Values

N/A

ShowAllColumns property

Data type

Logical

Description

Specifies whether the tableframe object expands horizontally at runtime to display all columns in the design.

Values

True, False

ShowGrid property

Data type

Logical

Description

Specifies whether the form or report grid is visible.

Values

True, False

Shrinkable property

Data type

Logical

Description

Specifies whether a report band can be shrunk.

Values

True, False

Size property

Data type

Point

Description

Specifies the coordinates of the lower-right corner of a design object, relative to its upper-left corner.

Values

N/A

SizeToFit property

Data type

Logical

Description

Specifies whether a form retains the size of the underlying page when opened. When the SizeToFit property is set to True, the form retains the size of the underlying page. When the SizeToFit property is set to False, the form opens at a preset, default size.

Values

True, False

SnapToGrid property

Data type

Logical

Description

Specifies whether object size and position are affected by the grid. When the SnapToGrid property is set to True, design objects jump to the closest minor division of the grid when moved or resized. Internally generated resizes (e.g., when you add text to a text object or define a field object) do not snap to the grid. When the SnapToGrid property is set to False, object size and position are not affected by the grid.

Values

True, False

SortOrder property

Data type

Logical

Description

Specifies a report's sort order. When the SortOrder property is set to True, the report is sorted in descending order. When the SortOrder property is set to False, the report is sorted in ascending order.

Values

Ascending, Descending

SpecialField property

Data type

SmallInt

Description

Determines a special field's type using SpecialFieldTypes constants. On the types that require a table, the field's current table is used. If there is no current table, the form or report's master table is used.

Values

DateField, NofFieldsField, NofPagesField, NofRecsField, PageNumField, RecordNoField, TableNameField, TimeField

SquareTabs property

Data type

Logical

Description

Specifies whether the tabs for the notebook pages have squared (Windows 95 style) rounded corners.

Values

True, False

StandardMenu property

Data type

Logical

Description

Specifies whether a form or report uses the standard Corel Paradox menus. When the StandardMenu property is set it to True, Corel Paradox menus are used. When the StandardMenu property is set it to False, the form or report uses alternate menus.

Values

True, False

StandardToolbar property

Data type

Logical

Description

Specifies whether a form or report uses the standard Toolbar. When the StandardToolbar property is set to True, the standard Toolbar is used. When the StandardToolbar property is set to False, the form or report uses an alternate toolbar.

Values

True, False

Start property

Data type

Point

Description

Specifies the coordinates of the start of a line. To specify the coordinates of the end of a line, see the End property.

Values

N/A

StartPageNumbers property

Data type

SmallInt

Description

Determines the starting value for a report's page numbers.

Values

>0

Style property

Data type

SmallInt

Description

Reports or specifies a button's display style.

Values

CorelButton, Windows3dButton, WindowsButton

SummaryModifier property

Data type

SmallInt

Description

Determines how to modify aggregator fields in reports. This property uses AggModifiers constants.

Values

CumulativeAgg, CumUniqueAgg, RegularAgg, UniqueAgg

TabHeight property

Data type

LongInt

Description

Specifies the height of the tab on the notebook page in twips.

TabsAcross property

Data type

SmallInt

Description

Specifies the maximum number of tabs that can appear across a notebook object. If the number of tabs is exceeds the TabsAcross value, and the HorizonatlScrollbar property is set to True, the tabs scroll across the notebook. If the number of tabs is exceeds the TabsAcross value, and the HorizonatlScrollbar property is set to False, the tabs are stacked in the notebook.

TabsOnTop property

Data type

Logical

Description

Specifies whether tabs are displayed at the top or bottom of a notebook.

Values

True, False

TabStop property

Data type

Logical

Description

Specifies whether a field object is a tab stop.

Values

True, False

TableName property

Data type

String

Description

Specifies the name of the table to which a design object is bound.

Values

N/A

Text property

Data type

String

Description

Specifies the characters that are displayed in a text object.

Values

N/A

ThickFrame property

Data type

Logical

Description

Specifies the thickness of a window frame. When the ThickFrame property is set to True, and DialogForm and Border also True, a thick frame replaces the usual pixel-wide frame.

Values

True, False

Thickness property

Data type

SmallInt

Description

Specifies a line's thickness.

Values

LWidth10Points, LWidth1Point, LWidth2Points, LWidth3Points, LWidth6Points, LWidthHairline, LWidthHalfPoint

Title property

Data type

String

Description

Specifies the text that is displayed in a form's caption.

Values

N/A

TopBorder property

Data type

LongInt

Description

Returns the size of an object's top border (in twips). You cannot embed another object within an object's border.

Values

N/A

TopLine property

Data type

LongInt

Description

The number of the line at the top of a text object.

Values

N/A

Touched property

Data type

Logical

Description

Specifies whether the user has made changes that have not yet been committed.

Values

True, False

Translucent property

Data type

Logical

Description

Specifies whether an object's color is translucent.

Values

True, False

UncheckedValue property

Data type

String

Description

Specifies the value that a disabled button writes to its parent field object. This property applies to check boxes only.

Values

N/A

Value property

Data type

String

Description

Specifies a design object's value.

Values

N/A

VerticalScrollBar property

Data type

Logical

Description

Specifies whether an object has a vertical scroll bar. This property is not valid for all UIObjects.

Values

True, False

Visible property

Data type

Logical

Description

Specifies whether an object is displayed in a form at run time. When the Visible property is set to True, it displays the object and the objects it contains. When the Visible property is set to False, it hides the object and the objects it contains.

Values

True, False

WideScrollBar property

Data type

Logical

Description

Determines whether a scroll bar is wide or narrow (the default).When the WiseScroolBar property is set to True, the scroll bar is wide; otherwise, the scrollbar is narrow.

Values

True, False

Width property

Data type

LongInt

Description

Determines the width (in twips) of a column in a Table window.

Values

>0

WordWrap property

Data type

Logical

Description

Specifies whether to wrap lines that exceed a field object's width.

Values

True, False

Xseparation property

Data type

LongInt

Description

Specifies the distance (in twips) between records in the indicated direction.

Values

>0

Yseparation property

Data type

LongInt

Description

Specifies the distance (in twips) between records in the indicated direction.

Values

>0

Properties unique to chart objects

Click one the following chart object properties for more information.

BackWall.Color
BackWall.Pattern.Color
BackWall.Pattern.Style
Background.Color
Background.Pattern.Color
Background.Pattern.Style
BaseFloor.Color
BaseFloor.Pattern.Color
BaseFloor.Pattern.Style
BindType
CurrentSeries
CurrentSlice
GraphType
Label.Font.Color
Label.Font.Script (8)
Label.Font.Size
Label.Font.Style
Label.Font.Typeface
Label.LabelFormat
Label.LabelLocation
Label.NumberFormat
LeftWall.Color
LeftWall.Pattern.Color
LeftWall.Pattern.Style
LegendBox.Color
LegendBox.Font.Color
LegendBox.Font.Script (8)
LegendBox.Font.Size
LegendBox.Font.Style
LegendBox.Font.Typeface
LegendBox.LegendPos
LegendBox.Pattern.Color
LegendBox.Pattern.Style
MaxGroups
MaxXValues
MinXValues
Options.Elevation
Options.Rotation
Options.ShowAxes
Options.ShowGrid
Options.ShowLabels
Options.ShowLegend
Options.ShowTitle
Series.Color
Series.Graph_Title.Font.Color
Series.Graph_Title.Font.Script (8)
Series.Graph_Title.Font.Size
Series.Graph_Title.Font.Style
Series.Graph_Title.Font.Typeface
Series.Graph_Title.Text
Series.Graph_Title.UseDefault
Series.Line.Color

Series.Line.LineStyle
Series.Line.Thickness
Series.Marker.Size
Series.Marker.Style
Series.Pattern.Color
Series.Pattern.Style
Series.TypeOverride
SeriesName
Slice.Color
Slice.Explode
Slice.Pattern.Color
Slice.Pattern.Style
TitleBox.Color
TitleBox.Graph_Title.Font.Color
TitleBox.Graph_Title.Font.Script (8)
TitleBox.Graph_Title.Font.Size
TitleBox.Graph_Title.Font.Style
TitleBox.Graph_Title.Font.Typeface
TitleBox.Graph_Title.Text
TitleBox.Graph_Title.UseDefault
TitleBox.Pattern.Color
TitleBox.Pattern.Style
TitleBox.Subtitle.Font.Color
TitleBox.Subtitle.Font.Script (8)
TitleBox.Subtitle.Font.Size
TitleBox.Subtitle.Font.Style
TitleBox.Subtitle.Font.Typeface
TitleBox.Subtitle.Text
TitleBox.Subtitle.UseDefault
TitleBoxName
XAxisName
XAxis.Graph_Title.Font.Color
XAxis.Graph_Title.Font.Script (8)
XAxis.Graph_Title.Font.Size
XAxis.Graph_Title.Font.Style
XAxis.Graph_Title.Font.Typeface
XAxis.Graph_Title.Text
XAxis.Graph_Title.UseDefault
XAxis.Scale.AutoScale
XAxis.Scale.HighValue
XAxis.Scale.Increment
XAxis.Scale.Logarithmic
XAxis.Scale.LowValue
XAxis.Ticks.Alternate
XAxis.Ticks.DateFormat
XAxis.Ticks.Font.Color
XAxis.Ticks.Font.Script (8)
XAxis.Ticks.Font.Size
XAxis.Ticks.Font.Style
XAxis.Ticks.Font.Typeface
XAxis.Ticks.NumberFormat
XAxis.Ticks.TimeFormat
XAxis.Ticks.TimeStampFormat
YAxisName
YAxis.Graph_Title.Font.Color
YAxis.Graph_Title.Font.Script (8)

YAxis.Graph_Title.Font.Size
YAxis.Graph_Title.Font.Style
YAxis.Graph_Title.Font.Typeface
YAxis.Graph_Title.Text
YAxis.Graph_Title.UseDefault
YAxis.Scale.AutoScale
YAxis.Scale.HighValue
YAxis.Scale.Increment
YAxis.Scale.Logarithmic
YAxis.Scale.LowValue
YAxis.Ticks.Alternate
YAxis.Ticks.DateFormat
YAxis.Ticks.Font.Color
YAxis.Ticks.Font.Script (8)
YAxis.Ticks.Font.Size
YAxis.Ticks.Font.Style
YAxis.Ticks.Font.Typeface
YAxis.Ticks.NumberFormat
YAxis.Ticks.TimeFormat
YAxis.Ticks.TimeStampFormat
ZAxisName
ZAxis.Graph_Title.Font.Color
ZAxis.Graph_Title.Font.Script (8)
ZAxis.Graph_Title.Font.Size
ZAxis.Graph_Title.Font.Style
ZAxis.Graph_Title.Font.Typeface
ZAxis.Graph_Title.Text
ZAxis.Graph_Title.UseDefault
ZAxis.Scale.AutoScale
ZAxis.Scale.HighValue
ZAxis.Scale.Increment
ZAxis.Scale.Logarithmic
ZAxis.Scale.LowValue
ZAxis.Ticks.Alternate
ZAxis.Ticks.DateFormat
ZAxis.Ticks.Font.Color
ZAxis.Ticks.Font.Script (8)
ZAxis.Ticks.Font.Size
ZAxis.Ticks.Font.Style
ZAxis.Ticks.Font.Typeface
ZAxis.Ticks.NumberFormat
ZAxis.Ticks.TimeFormat
ZAxis.Ticks.TimeStampFormat

{button ,AL(`OBJECT_PROPERTIES;',0,"Defaultoverview",)} **Related Topics**

BackWall.Color

This property applies only to chart objects.

Data type

LongInt

Values

Black, Blue, Brown, DarkBlue, DarkCyan, DarkGray, DarkGreen, DarkMagenta, DarkRed, Gray, Green, LightBlue, Magenta, Red, White, Yellow, Transparent

BackWall.Pattern.Color

This property applies only to chart objects.

Data type

LongInt

Values

Black, Blue, Brown, DarkBlue, DarkCyan, DarkGray, DarkGreen, DarkMagenta, DarkRed, Gray, Green, LightBlue, Magenta, Red, White, Yellow, Transparent

BackWall.Pattern.Style

This property applies only to chart objects.

Data type

SmallInt

Values

BricksPattern, CrosshatchPattern, DiagonalCrosshatchPattern, DottedLinePattern, EmptyPattern, FuzzyStripesDownPattern, HeavyDotPattern, HorizontalLinesPattern, LatticePattern, LeftDiagonalLinesPattern, LightDotPattern, MaximumDotPattern, MediumDotPattern, RightDiagonalLinesPattern, ScalesPattern, StaggeredDashesPattern, ThickHorizontalLinesPattern, ThickStripesDownPattern, ThickStripesUpPattern, ThickVerticalLinesPattern, VerticalLinesPattern, VeryHeavyDotPattern, WeavePattern, ZigZagPattern

Background.Color

This property applies only to chart objects.

Data type

LongInt

Values

Black, Blue, Brown, DarkBlue, DarkCyan, DarkGray, DarkGreen, DarkMagenta, DarkRed, Gray, Green, LightBlue, Magenta, Red, White, Yellow, Transparent

Background.Pattern.Color

This property applies only to chart objects.

Data type

LongInt

Values

Black, Blue, Brown, DarkBlue, DarkCyan, DarkGray, DarkGreen, DarkMagenta, DarkRed, Gray, Green, LightBlue, Magenta, Red, White, Yellow, Transparent

Background.Pattern.Style

This property applies only to chart objects.

Data type

SmallInt

Values

BricksPattern, CrosshatchPattern, DiagonalCrosshatchPattern, DottedLinePattern, EmptyPattern, FuzzyStripesDownPattern, HeavyDotPattern, HorizontalLinesPattern, LatticePattern, LeftDiagonalLinesPattern, LightDotPattern, MaximumDotPattern, MediumDotPattern, RightDiagonalLinesPattern, ScalesPattern, StaggeredDashesPattern, ThickHorizontalLinesPattern, ThickStripesDownPattern, ThickStripesUpPattern, ThickVerticalLinesPattern, VerticalLinesPattern, VeryHeavyDotPattern, WeavePattern, ZigZagPattern

BaseFloor.Color

This property applies only to chart objects.

Data type

LongInt

Values

Black, Blue, Brown, DarkBlue, DarkCyan, DarkGray, DarkGreen, DarkMagenta, DarkRed, Gray, Green, LightBlue, Magenta, Red, White, Yellow, Transparent

BaseFloor.Pattern.Color

This property applies only to chart objects.

Data type

LongInt

Values

Black, Blue, Brown, DarkBlue, DarkCyan, DarkGray, DarkGreen, DarkMagenta, DarkRed, Gray, Green, LightBlue, Magenta, Red, White, Yellow, Transparent

BaseFloor.Pattern.Style

This property applies only to chart objects.

Data type

SmallInt

Values

BricksPattern, CrosshatchPattern, DiagonalCrosshatchPattern, DottedLinePattern, EmptyPattern, FuzzyStripesDownPattern, HeavyDotPattern, HorizontalLinesPattern, LatticePattern, LeftDiagonalLinesPattern, LightDotPattern, MaximumDotPattern, MediumDotPattern, RightDiagonalLinesPattern, ScalesPattern, StaggeredDashesPattern, ThickHorizontalLinesPattern, ThickStripesDownPattern, ThickStripesUpPattern, ThickVerticalLinesPattern, VerticalLinesPattern, VeryHeavyDotPattern, WeavePattern, ZigZagPattern

BindType

This property applies only to chart objects.

Data type

SmallInt

Values

Graph1DSummary, Graph2DSummary, GraphTabular

CurrentSeries

This property applies only to chart objects.

Data type

SmallInt

Values

N/A

CurrentSlice

This property applies only to chart objects.

Data type

SmallInt

Values

N/A

GraphType

This property applies only to chart objects.

Data type

SmallInt

Values

Graph2DArea, Graph2DBar, Graph2DColumns, Graph2DLine, Graph2DPie, Graph2DRotatedBar, Graph2DStackedBar, Graph3DArea, Graph3DBar, Graph3DColumns, Graph3DPie, Graph3DRibbon, Graph3DRotatedBar, Graph3DStackedBar, Graph3DStep, Graph3DSurface, GraphXY

Label.Font.Color

This property applies only to chart objects.

Data type

LongInt

Values

Black, Blue, Brown, DarkBlue, DarkCyan, DarkGray, DarkGreen, DarkMagenta, DarkRed, Gray, Green, LightBlue, Magenta, Red, White, Yellow, Transparent

Label.Font.Script

The language script or character set of the chosen font. This property applies only to chart objects.

Data type

String

Values

Depends on the installed fonts.

Label.Font.Size

This property applies only to chart objects.

Data type

SmallInt

Values

Depends on system

Label.Font.Style

This property applies only to chart objects.

Data type

SmallInt

Values

FontAttribBold, FontAttribItalic, FontAttribNormal, FontAttribStrikeout, FontAttribUnderline

Label.Font.Typeface

This property applies only to chart objects.

Data type

String

Values

Depends on system

Label.LabelFormat

This property applies only to chart objects.

Data type

SmallInt

Values

GraphHideY, GraphPercent, GraphShowY

Label.LabelLocation

This property applies only to chart objects.

Data type

SmallInt

Values

Above, Below, Bottom, Center, Left, Middle, Right, Top

Label.NumberFormat

This property applies only to chart objects.

Data type

N/A

Values

Format specification

LeftWall.Color

This property applies only to chart objects.

Data type

LongInt

Values

Black, Blue, Brown, DarkBlue, DarkCyan, DarkGray, DarkGreen, DarkMagenta, DarkRed, Gray, Green, LightBlue, Magenta, Red, White, Yellow, Transparent

LeftWall.Pattern.Color

This property applies only to chart objects.

Data type

LongInt

Values

Black, Blue, Brown, DarkBlue, DarkCyan, DarkGray, DarkGreen, DarkMagenta, DarkRed, Gray, Green, LightBlue, Magenta, Red, White, Yellow, Transparent

LeftWall.Pattern.Style

This property applies only to chart objects.

Data type

SmallInt

Values

BricksPattern, CrosshatchPattern, DiagonalCrosshatchPattern, DottedLinePattern, EmptyPattern, FuzzyStripesDownPattern, HeavyDotPattern, HorizontalLinesPattern, LatticePattern, LeftDiagonalLinesPattern, LightDotPattern, MaximumDotPattern, MediumDotPattern, RightDiagonalLinesPattern, ScalesPattern, StaggeredDashesPattern, ThickHorizontalLinesPattern, ThickStripesDownPattern, ThickStripesUpPattern, ThickVerticalLinesPattern, VerticalLinesPattern, VeryHeavyDotPattern, WeavePattern, ZigZagPattern

LegendBox.Color

This property applies only to chart objects.

Data type

LongInt

Values

Black, Blue, Brown, DarkBlue, DarkCyan, DarkGray, DarkGreen, DarkMagenta, DarkRed, Gray, Green, LightBlue, Magenta, Red, White, Yellow, Transparent

LegendBox.Font.Color

This property applies only to chart objects.

Data type

LongInt

Values

Black, Blue, Brown, DarkBlue, DarkCyan, DarkGray, DarkGreen, DarkMagenta, DarkRed, Gray, Green, LightBlue, Magenta, Red, White, Yellow, Transparent

LegendBox.Font.Script

The language script or character set of the chosen font. This property applies only to chart objects.

Data type

String

Values

Depends on the installed fonts.

LegendBox.Font.Size

This property applies only to chart objects.

Data type

SmallInt

Values

Depends on system

LegendBox.Font.Style

This property applies only to chart objects.

Data type

SmallInt

Values

FontAttribBold, FontAttribItalic, FontAttribNormal, FontAttribStrikeout, FontAttribUnderline

LegendBox.Font.Typeface

This property applies only to chart objects.

Data type

String

Values

Depends on system

LegendBox.LegendPos

This property applies only to chart objects.

Data type

SmallInt

Values

Bottom, Right

LegendBox.Pattern.Color

This property applies only to chart objects.

Data type

LongInt

Values

Black, Blue, Brown, DarkBlue, DarkCyan, DarkGray, DarkGreen, DarkMagenta, DarkRed, Gray, Green, LightBlue, Magenta, Red, White, Yellow, Transparent

LegendBox.Pattern.Style

This property applies only to chart objects.

Data type

SmallInt

Values

BricksPattern, CrosshatchPattern, DiagonalCrosshatchPattern, DottedLinePattern, EmptyPattern, FuzzyStripesDownPattern, HeavyDotPattern, HorizontalLinesPattern, LatticePattern, LeftDiagonalLinesPattern, LightDotPattern, MaximumDotPattern, MediumDotPattern, RightDiagonalLinesPattern, ScalesPattern, StaggeredDashesPattern, ThickHorizontalLinesPattern, ThickStripesDownPattern, ThickStripesUpPattern, ThickVerticalLinesPattern, VerticalLinesPattern, VeryHeavyDotPattern, WeavePattern, ZigZagPattern

MaxGroups

This property applies only to chart objects.

Data type

SmallInt

Values

Depends on chart

MaxXValues

This property applies only to chart objects.

Data type

SmallInt

Values

Depends on chart

MinXValues

This property applies only to chart objects.

Data type

SmallInt

Values

Depends on chart

Options.Elevation

This property applies only to chart objects.

Data type

SmallInt

Values

0 to 90 degrees

Options.Rotation

This property applies only to chart objects.

Data type

SmallInt

Values

0 to 90 degrees

Options.ShowAxes

This property applies only to chart objects.

Data type

Logical

Values

True, False

Options.ShowGrid

This property applies only to chart objects.

Data type

Logical

Values

True, False

Options.ShowLabels

This property applies only to chart objects.

Data type

Logical

Values

True, False

Options.ShowLegend

This property applies only to chart objects.

Data type

Logical

Values

True, False

Options.ShowTitle

This property applies only to chart objects.

Data type

Logical

Values

True, False

Series.Color

This property applies only to chart objects.

Data type

LongInt

Values

Black, Blue, Brown, DarkBlue, DarkCyan, DarkGray, DarkGreen, DarkMagenta, DarkRed, Gray, Green, LightBlue, Magenta, Red, White, Yellow, Transparent

Series.Graph_Title.Font.Color

This property applies only to chart objects.

Data type

LongInt

Values

Black, Blue, Brown, DarkBlue, DarkCyan, DarkGray, DarkGreen, DarkMagenta, DarkRed, Gray, Green, LightBlue, Magenta, Red, White, Yellow, Transparent

Series.Graph_Title.Font.Script

The language script or character set of the chosen font. This property applies only to chart objects.

Data type

String

Values

Depends on the installed fonts.

Series.Graph_Title.Font.Size

This property applies only to chart objects.

Data type

SmallInt

Values

Depends on system

Series.Graph_Title.Font.Style

This property applies only to chart objects.

Data type

SmallInt

Values

FontAttribBold, FontAttribItalic, FontAttribNormal, FontAttribStrikeout, FontAttribUnderline

Series.Graph_Title.Font.Typeface

This property applies only to chart objects.

Data type

String

Values

Depends on system

Series.Graph_Title.Text

This property applies only to chart objects.

Data type

String

Values

N/A

Series.Graph_Title.UseDefault

This property applies only to chart objects.

Data type

Logical

Values

True, False

Series.Line.Color

This property applies only to chart objects.

Data type

LongInt

Values

Black, Blue, Brown, DarkBlue, DarkCyan, DarkGray, DarkGreen, DarkMagenta, DarkRed, Gray, Green, LightBlue, Magenta, Red, White, Yellow, Transparent

Series.Line.LineStyle

This property applies only to chart objects.

Data type

SmallInt

Values

DashDotDotLine, DashDotLine, DashedLine, DottedLine, NoLine, SolidLine

Series.Line.Thickness

This property applies only to chart objects.

Data type

SmallInt

Values

LWidth10Points, LWidth1Point, LWidth2Points, LWidth3Points, LWidth6Points, LWidthHairline, LWidthHalfPoint

Series.Marker.Size

This property applies only to chart objects.

Data type

SmallInt

Values

MarkerSize0, MarkerSize2, MarkerSize4, MarkerSize8, MarkerSize12, MarkerSize18, MarkerSize24, MarkerSize36, MarkerSize54, MarkerSize72

Series.Marker.Style

This property applies only to chart objects.

Data type

SmallInt

Values

MarkerBoxedCross, MarkerBoxed_Plus, MarkerCross, MarkerFilledBox, MarkerFilledCircle, MarkerFilledDownTriangle, MarkerFilledTriangle, MarkerFilledTriangles, MarkerHollowBox, MarkerHollowCircle, MarkerHollowDownTriangle, MarkerHollowTriangle, MarkerHollowTriangles, MarkerHorizontalLine, MarkerPlus, MarkerVerticalLine

Series.Pattern.Color

This property applies only to chart objects.

Data type

LongInt

Values

Black, Blue, Brown, DarkBlue, DarkCyan, DarkGray, DarkGreen, DarkMagenta, DarkRed, Gray, Green, LightBlue, Magenta, Red, White, Yellow, Transparent

Series.Pattern.Style

This property applies only to chart objects.

Data type

SmallInt

Values

BricksPattern, CrosshatchPattern, DiagonalCrosshatchPattern, DottedLinePattern, EmptyPattern, FuzzyStripesDownPattern, HeavyDotPattern, HorizontalLinesPattern, LatticePattern, LeftDiagonalLinesPattern, LightDotPattern, MaximumDotPattern, MediumDotPattern, RightDiagonalLinesPattern, ScalesPattern, StaggeredDashesPattern, ThickHorizontalLinesPattern, ThickStripesDownPattern, ThickStripesUpPattern, ThickVerticalLinesPattern, VerticalLinesPattern, VeryHeavyDotPattern, WeavePattern, ZigZagPattern

Series.TypeOverride

This property applies only to chart objects.

Data type

SmallInt

Values

Graph2DArea, Graph2DBar, Graph2DLine, None

SeriesName

This property applies only to chart objects.

Data type

String

Values

N/A

Slice.Color

This property applies only to chart objects.

Data type

LongInt

Values

Black, Blue, Brown, DarkBlue, DarkCyan, DarkGray, DarkGreen, DarkMagenta, DarkRed, Gray, Green, LightBlue, Magenta, Red, White, Yellow, Transparent

Slice.Explode

This property applies only to chart objects.

Data type

Logical

Values

True, False

Slice.Pattern.Color

This property applies only to chart objects.

Data type

LongInt

Values

Black, Blue, Brown, DarkBlue, DarkCyan, DarkGray, DarkGreen, DarkMagenta, DarkRed, Gray, Green, LightBlue, Magenta, Red, White, Yellow, Transparent

Slice.Pattern.Style

This property applies only to chart objects.

Data type

SmallInt

Values

BricksPattern, CrosshatchPattern, DiagonalCrosshatchPattern, DottedLinePattern, EmptyPattern, FuzzyStripesDownPattern, HeavyDotPattern, HorizontalLinesPattern, LatticePattern, LeftDiagonalLinesPattern, LightDotPattern, MaximumDotPattern, MediumDotPattern, RightDiagonalLinesPattern, ScalesPattern, StaggeredDashesPattern, ThickHorizontalLinesPattern, ThickStripesDownPattern, ThickStripesUpPattern, ThickVerticalLinesPattern, VerticalLinesPattern, VeryHeavyDotPattern, WeavePattern, ZigZagPattern

TitleBox.Color

This property applies only to chart objects.

Data type

LongInt

Values

Black, Blue, Brown, DarkBlue, DarkCyan, DarkGray, DarkGreen, DarkMagenta, DarkRed, Gray, Green, LightBlue, Magenta, Red, White, Yellow, Transparent

TitleBox.Graph_Title.Font.Color

This property applies only to chart objects.

Data type

LongInt

Values

Black, Blue, Brown, DarkBlue, DarkCyan, DarkGray, DarkGreen, DarkMagenta, DarkRed, Gray, Green, LightBlue, Magenta, Red, White, Yellow, Transparent

TitleBox.Graph_Title.Font.Script

The language script or character set of the chosen font. This property applies only to chart objects.

Data type

String

Values

Depends on the installed fonts.

TitleBox.Graph_Title.Font.Size

This property applies only to chart objects.

Data type

SmallInt

Values

Depends on system

TitleBox.Graph_Title.Font.Style

This property applies only to chart objects.

Data type

SmallInt

Values

FontAttribBold, FontAttribItalic, FontAttribNormal, FontAttribStrikeout, FontAttribUnderline

TitleBox.Graph_Title.Font.Typeface

This property applies only to chart objects.

Data type

String

Values

Depends on system

TitleBox.Graph_Title.Text

This property applies only to chart objects.

Data type

String

Values

N/A

TitleBox.Graph_Title.UseDefault

This property applies only to chart objects.

Data type

Logical

Values

True, False

TitleBox.Pattern.Color

This property applies only to chart objects.

Data type

LongInt

Values

Black, Blue, Brown, DarkBlue, DarkCyan, DarkGray, DarkGreen, DarkMagenta, DarkRed, Gray, Green, LightBlue, Magenta, Red, White, Yellow, Transparent

TitleBox.Pattern.Style

This property applies only to chart objects.

Data type

SmallInt

Values

BricksPattern, CrosshatchPattern, DiagonalCrosshatchPattern, DottedLinePattern, EmptyPattern, FuzzyStripesDownPattern, HeavyDotPattern, HorizontalLinesPattern, LatticePattern, LeftDiagonalLinesPattern, LightDotPattern, MaximumDotPattern, MediumDotPattern, RightDiagonalLinesPattern, ScalesPattern, StaggeredDashesPattern, ThickHorizontalLinesPattern, ThickStripesDownPattern, ThickStripesUpPattern, ThickVerticalLinesPattern, VerticalLinesPattern, VeryHeavyDotPattern, WeavePattern, ZigZagPattern

TitleBox.Subtitle.Font.Color

This property applies only to chart objects.

Data type

LongInt

Values

Black, Blue, Brown, DarkBlue, DarkCyan, DarkGray, DarkGreen, DarkMagenta, DarkRed, Gray, Green, LightBlue, Magenta, Red, White, Yellow, Transparent

TitleBox.Subtitle.Font.Script

The language script or character set of the chosen font. This property applies only to chart objects.

Data type

String

Values

Depends on the installed fonts.

TitleBox.Subtitle.Font.Size

This property applies only to chart objects.

Data type

SmallInt

Values

Depends on system

TitleBox.Subtitle.Font.Style

This property applies only to chart objects.

Data type

SmallInt

Values

FontAttribBold, FontAttribItalic, FontAttribNormal, FontAttribStrikeout, FontAttribUnderline

TitleBox.Subtitle.Font.Typeface

This property applies only to chart objects.

Data type

String

Values

Depends on system

TitleBox.Subtitle.Text

This property applies only to chart objects.

Data type

String

Values

N/A

TitleBox.Subtitle.UseDefault

This property applies only to chart objects.

Data type

Logical

Values

True, False

TitleBoxName

This property applies only to chart objects.

Data type

String

Values

N/A

XAxisName

This property applies only to chart objects.

Data type

String

Values

N/A

XAxis.Graph_Title.Font.Color

This property applies only to chart objects.

Data type

LongInt

Values

Black, Blue, Brown, DarkBlue, DarkCyan, DarkGray, DarkGreen, DarkMagenta, DarkRed, Gray, Green, LightBlue, Magenta, Red, White, Yellow, Transparent

XAxis.Graph_Title.Font.Script

The language script or character set of the chosen font. This property applies only to chart objects.

Data type

String

Values

Depends on the installed fonts.

XAxis.Graph_Title.Font.Size

This property applies only to chart objects.

Data type

SmallInt

Values

Depends on system

XAxis.Graph_Title.Font.Style

This property applies only to chart objects.

Data type

SmallInt

Values

FontAttribBold, FontAttribItalic, FontAttribNormal, FontAttribStrikeout, FontAttribUnderline

XAxis.Graph_Title.Font.Typeface

This property applies only to chart objects.

Data type

String

Values

Depends on system

XAxis.Graph_Title.Text

This property applies only to chart objects.

Data type

String

Values

N/A

XAxis.Graph_Title.UseDefault

This property applies only to chart objects.

Data type

Logical

Values

True, False

XAxis.Scale.AutoScale

This property applies only to chart objects.

Data type

Logical

Values

True, False

XAxis.Scale.HighValue

This property applies only to chart objects.

Data type

Number

Values

Depends on chart

XAxis.Scale.Increment

This property applies only to chart objects.

Data type

Number

Values

Depends on chart

XAxis.Scale.Logarithmic

This property applies only to chart objects.

Data type

Logical

Values

True, False

XAxis.Scale.LowValue

This property applies only to chart objects.

Data type

Number

Values

Depends on chart

XAxis.Ticks.Alternate

This property applies only to chart objects.

Data type

Logical

Values

True, False

XAxis.Ticks.DateFormat

This property applies only to chart objects.

Data type

N/A

Values

Format specification

XAxis.Ticks.Font.Color

This property applies only to chart objects.

Data type

LongInt

Values

Black, Blue, Brown, DarkBlue, DarkCyan, DarkGray, DarkGreen, DarkMagenta, DarkRed, Gray, Green, LightBlue, Magenta, Red, White, Yellow, Transparent

XAxis.Ticks.Font.Script

The language script or character set of the chosen font. This property applies only to chart objects.

Data type

String

Values

Depends on the installed fonts.

XAxis.Ticks.Font.Size

This property applies only to chart objects.

Data type

SmallInt

Values

Depends on system

XAxis.Ticks.Font.Style

This property applies only to chart objects.

Data type

SmallInt

Values

FontAttribBold, FontAttribItalic, FontAttribNormal, FontAttribStrikeout, FontAttribUnderline

XAxis.Ticks.Font.Typeface

This property applies only to chart objects.

Data type

String

Values

Depends on system

XAxis.Ticks.NumberFormat

This property applies only to chart objects.

Data type

N/A

Values

Format specification

XAxis.Ticks.TimeFormat

This property applies only to chart objects.

Data type

String

Values

N/A

XAxis.Ticks.TimeStampFormat

This property applies only to chart objects.

Data type

String

Values

N/A

YAxisName

This property applies only to chart objects.

Data type

String

Values

N/A

YAxis.Graph_Title.Font.Color

This property applies only to chart objects.

Data type

LongInt

Values

Black, Blue, Brown, DarkBlue, DarkCyan, DarkGray, DarkGreen, DarkMagenta, DarkRed, Gray, Green, LightBlue, Magenta, Red, White, Yellow, Transparent

YAxis.Graph_Title.Font.Script

The language script or character set of the chosen font. This property applies only to chart objects.

Data type

String

Values

Depends on the installed fonts.

YAxis.Graph_Title.Font.Size

This property applies only to chart objects.

Data type

SmallInt

Values

Depends on system

YAxis.Graph_Title.Font.Style

This property applies only to chart objects.

Data type

SmallInt

Values

FontAttribBold, FontAttribItalic, FontAttribNormal, FontAttribStrikeout, FontAttribUnderline

YAxis.Graph_Title.Font.Typeface

This property applies only to chart objects.

Data type

String

Values

Depends on system

YAxis.Graph_Title.Text

This property applies only to chart objects.

Data type

String

Values

N/A

YAxis.Graph_Title.UseDefault

This property applies only to chart objects.

Data type

Logical

Values

True, False

YAxis.Scale.AutoScale

This property applies only to chart objects.

Data type

Logical

Values

True, False

YAxis.Scale.HighValue

This property applies only to chart objects.

Data type

Number

Values

Depends on chart

YAxis.Scale.Increment

This property applies only to chart objects.

Data type

Number

Values

Depends on chart

YAxis.Scale.Logarithmic

This property applies only to chart objects.

Data type

Logical

Values

True, False

YAxis.Scale.LowValue

This property applies only to chart objects.

Data type

Number

Values

Depends on chart

YAxis.Ticks.Alternate

This property applies only to chart objects.

Data type

Logical

Values

True, False

YAxis.Ticks.DateFormat

This property applies only to chart objects.

Data type

N/A

Values

Format specification

YAxis.Ticks.Font.Color

This property applies only to chart objects.

Data type

LongInt

Values

Black, Blue, Brown, DarkBlue, DarkCyan, DarkGray, DarkGreen, DarkMagenta, DarkRed, Gray, Green, LightBlue, Magenta, Red, White, Yellow, Transparent

YAxis.Ticks.Font.Script

The language script or character set of the chosen font. This property applies only to chart objects.

Data type

String

Values

Depends on the installed fonts.

YAxis.Ticks.Font.Size

This property applies only to chart objects.

Data type

SmallInt

Values

Depends on system

YAxis.Ticks.Font.Style

This property applies only to chart objects.

Data type

SmallInt

Values

FontAttribBold, FontAttribItalic, FontAttribNormal, FontAttribStrikeout, FontAttribUnderline

YAxis.Ticks.Font.Typeface

This property applies only to chart objects.

Data type

String

Values

Depends on system

YAxis.Ticks.NumberFormat

This property applies only to chart objects.

Data type

N/A

Values

Format specification

YAxis.Ticks.TimeFormat

This property applies only to chart objects.

Data type

String

Values

N/A

YAxis.Ticks.TimeStampFormat

This property applies only to chart objects.

Data type

String

Values

N/A

ZAxisName

This property applies only to chart objects.

Data type

String

Values

N/A

ZAxis.Graph_Title.Font.Color

This property applies only to chart objects.

Data type

LongInt

Values

Black, Blue, Brown, DarkBlue, DarkCyan, DarkGray, DarkGreen, DarkMagenta, DarkRed, Gray, Green, LightBlue, Magenta, Red, White, Yellow, Transparent

ZAxis.Graph_Title.Font.Script

The language script or character set of the chosen font. This property applies only to chart objects.

Data type

String

Values

Depends on the installed fonts.

ZAxis.Graph_Title.Font.Size

This property applies only to chart objects.

Data type

SmallInt

Values

Depends on system

ZAxis.Graph_Title.Font.Style

This property applies only to chart objects.

Data type

SmallInt

Values

FontAttribBold, FontAttribItalic, FontAttribNormal, FontAttribStrikeout, FontAttribUnderline

ZAxis.Graph_Title.Font.Typeface

This property applies only to chart objects.

Data type

String

Values

Depends on system

ZAxis.Graph_Title.Text

This property applies only to chart objects.

Data type

String

Values

N/A

ZAxis.Graph_Title.UseDefault

This property applies only to chart objects.

Data type

Logical

Values

True, False

ZAxis.Scale.AutoScale

This property applies only to chart objects.

Data type

Logical

Values

True, False

ZAxis.Scale.HighValue

This property applies only to chart objects.

Data type

Number

Values

Depends on chart

ZAxis.Scale.Increment

This property applies only to chart objects.

Data type

Number

Values

Depends on chart

ZAxis.Scale.Logarithmic

This property applies only to chart objects.

Data type

Logical

Values

True, False

ZAxis.Scale.LowValue

This property applies only to chart objects.

Data type

Number

Values

Depends on chart

ZAxis.Ticks.Alternate

This property applies only to chart objects.

Data type

Logical

Values

True, False

ZAxis.Ticks.DateFormat

This property applies only to chart objects.

Data type

String

Values

N/A

ZAxis.Ticks.Font.Color

This property applies only to chart objects.

Data type

LongInt

Values

Black, Blue, Brown, DarkBlue, DarkCyan, DarkGray, DarkGreen, DarkMagenta, DarkRed, Gray, Green, LightBlue, Magenta, Red, White, Yellow, Transparent

ZAxis.Ticks.Font.Script

The language script or character set of the chosen font. This property applies only to chart objects.

Data type

String

Values

Depends on the installed fonts.

ZAxis.Ticks.Font.Size

This property applies only to chart objects.

Data type

SmallInt

Values

Depends on system

ZAxis.Ticks.Font.Style

This property applies only to chart objects.

Data type

SmallInt

Values

FontAttribBold, FontAttribItalic, FontAttribNormal, FontAttribStrikeout, FontAttribUnderline

ZAxis.Ticks.Font.Typeface

This property applies only to chart objects.

Data type

String

Values

Depends on system

ZAxis.Ticks.NumberFormat

This property applies only to chart objects.

Data type

String

Values

N/A

ZAxis.Ticks.TimeFormat

This property applies only to chart objects.

Data type

String

Values

N/A

ZAxis.Ticks.TimeStampFormat

This property applies only to chart objects.

Data type

String

Values

N/A

UIObject properties

Choose from the following UIObjects for a list of applicable properties.

Band
Bitmap
Box
Button
Cell
Chart
Crosstab
EditRegion
Ellipse
Field
Form
Graph (see Chart)
Group
Header
Line
List
Multirecord
Notebook
NotebookPage
OLE
Page
Record
TableFrame
TableView
Text
TVData
TVHeading

{button ,AL(`OBJECT_PROPERTIES;',0,"Defaultoverview",)} Related Topics

Band properties

Arrived (read-only)
BottomBorder (read-only) (5.0)
Breakable
Class (read-only)
ContainerName (read-only)
DefineGroup
DrillDown (8)
Enabled (7)
FieldName
First (read-only)
Focus (read-only)
FullName (read-only)
FullSize (read-only)
GroupRecords (5.0)
Headings
LeftBorder (read-only) (5.0)
Manager (read-only)
Name
Next (read-only)
OtherBandName (read-only) (5.0)
Owner (read-only)
Position
PositionalOrder (5.0)
PrecedePageHeader
Prev (read-only)
PrintOn1stPage
Range (5.0)
RightBorder (read-only) (5.0)
Scroll
Select (4.5)
Shrinkable
Size
SortOrder
StartPageNumbers (read-only) (5.0)
TopBorder (read-only) (5.0)

Bitmap properties

Arrived (read-only)
BottomBorder (read-only) (5.0)
Class (read-only)
ContainerName (read-only)
Design.ContainObjects
Design.PinHorizontal
Design.PinVertical
Design.Selectable (4.5)
Design.SizeToFit
Enabled (7)
First (read-only)
Focus (read-only)
Frame.Color
Frame.Style
Frame.Thickness
FullName (read-only)
FullSize (read-only)
HorizontalScrollBar
LeftBorder (read-only) (5.0)
Magnification
Manager (read-only)
Name
Next (read-only)
NextTabStop (5.0)
Owner (read-only)
Position
Prev (read-only)
RasterOperation
RightBorder (read-only) (5.0)
Scroll
Select (4.5)
Size
TopBorder (read-only) (5.0)
Value
VerticalScrollBar
Visible

Box properties

Arrived (read-only)
BottomBorder (read-only) (5.0)
Breakable
Class (read-only)
Color
ContainerName (read-only)
Design.ContainObjects
Design.PinHorizontal
Design.PinVertical
Design.Selectable (4.5)
Enabled (7)
First (read-only)
Focus (read-only)
Frame.Color
Frame.Style
Frame.Thickness
FullName (read-only)
FullSize (read-only)
Invisible (5.0)
LeftBorder (read-only) (5.0)
Manager (read-only)
Name
Next (read-only)
Owner (read-only)
Pattern.Color
Pattern.Style
Position
Prev (read-only)
RightBorder (read-only) (5.0)
Select (4.5)
Shrinkable
Size
TopBorder (read-only) (5.0)
Translucent
Visible

Button properties

Arrived (read-only)
BottomBorder (read-only) (5.0)
ButtonType
CenterLabel
CheckedValue (5.0)
Class (read-only)
ContainerName (read-only)
Design.ContainObjects
Design.PinHorizontal
Design.PinVertical
Design.Selectable (4.5)
Enabled (7)
First (read-only)
FitHeight
FitWidth
FlatLook
Focus (read-only)
FullName (read-only)
FullSize (read-only)
LabelText
LeftBorder (read-only) (5.0)
Manager (read-only)
Name
Next (read-only)
NextTabStop (5.0)
Owner (read-only)
Position
Prev (read-only)
RightBorder (read-only) (5.0)
Select (4.5)
Size
Style
TabStop
TopBorder (read-only) (5.0)
UncheckedValue (5.0)
Value
Visible

Cell properties

Arrived (read-only)
BottomBorder (read-only) (5.0)
Class (read-only)
Color
ContainerName (read-only)
Design.ContainObjects
Design.PinHorizontal
Design.PinVertical
Design.Selectable (4.5)
Design.SizeToFit
Enabled (7)
First (read-only)
FitWidth
Focus (read-only)
FullName (read-only)
FullSize (read-only)
HorizontalScrollBar
LeftBorder (read-only) (5.0)
Manager (read-only)
Name
Next (read-only)
NextTabStop (5.0)
Owner (read-only)
Position
Prev (read-only)
RightBorder (read-only) (5.0)
Scroll
Select (4.5)
Size
TopBorder (read-only) (5.0)
VerticalScrollBar

Crosstab properties

Arrived (read-only)
Breakable
Class (read-only)
Color
ContainerName (read-only)
CurrentColumn (5.0)
CurrentRow (5.0)
Design.ContainObjects
Design.PinHorizontal
Design.PinVertical
Design.Selectable (4.5)
Design.SizeToFit
Enabled (7)
First (read-only)
FitHeight
FitWidth
Focus (read-only)
FullName (read-only)
FullSize (read-only)
Grid.Color
Grid.GridStyle
HorizontalScrollBar
Manager (read-only)
Name
Next (read-only)
NextTabStop (5.0)
Owner (read-only)
PinHorizontal
PinVertical
Position
Prev (read-only)
Scroll
Select (4.5)
Size
TableName
Touched (read-only)
Translucent
VerticalScrollBar
Visible

EditRegion properties

Alignment
Arrived (read-only)
AvgCharSize (read-only) (5.0)
BottomBorder (read-only) (5.0)
Breakable
Class (read-only)
Color
CompleteDisplay
ContainerName (read-only)
Design.ContainObjects
Design.PinHorizontal
Design.PinVertical
Design.Selectable (4.5)
Design.SizeToFit
DisplayType (read-only)
Enabled (7)
First (read-only)
FitHeight
FitWidth
Focus (read-only)
Font.Color
Font.Script (8)
Font.Size
Font.Style
Font.Typeface
Format.DateFormat
Format.LogicalFormat
Format.NumberFormat
Format.TimeFormat
Format.TimeStampFormat
Frame.Color
Frame.Style
Frame.Thickness
FullName (read-only)
FullSize (read-only)
LeftBorder (read-only) (5.0)
Magnification
Manager (read-only)
Name
Next (read-only)
NextTabStop (5.0)
NoEcho
Owner (read-only)
Position
Prev (read-only)
RasterOperation
Readonly
RightBorder (read-only) (5.0)
Scroll
Select (4.5)
Size
TabStop
Text

TopBorder (read-only) (5.0)
Translucent
Value
Visible
WordWrap

Ellipse properties

Arrived (read-only)
BottomBorder (read-only) (5.0)
Class (read-only)
Color
ContainerName (read-only)
Design.ContainObjects
Design.PinHorizontal
Design.PinVertical
Design.Selectable (4.5)
Enabled (7)
First (read-only)
Focus (read-only)
FullName (read-only)
FullSize (read-only)
LeftBorder (read-only) (5.0)
Line.Color
Line.LineStyle
Line.Thickness
Manager (read-only)
Name
Next (read-only)
NextTabStop (5.0)
Owner (read-only)
Pattern.Color
Pattern.Style
Position
Prev (read-only)
RightBorder (read-only) (5.0)
Select (4.5)
Shrinkable
Size
TopBorder (read-only) (5.0)
Translucent
Visible

Field properties

Alignment
Arrived (read-only)
AutoAppend (4.5)
AvgCharSize (read-only) (5.0)
BlankRecord (read-only)
BottomBorder (read-only) (5.0)
Breakable
CalculatedField (5.0)
Class (read-only)
Color
CompleteDisplay
ContainerName (read-only)
CursorColumn
CursorLine
CursorPos
Default (read-only)
Deleted (read-only)
DeleteWhenEmpty (5.0)
Design.ContainObjects
Design.PinHorizontal
Design.PinVertical
Design.Selectable (4.5)
Design.SizeToFit
DisplayType
Editing (read-only) (4.5)
Enabled (7)
FieldName
FieldNo (read-only)
FieldRights (read-only)
FieldSize (read-only)
FieldType (read-only)
FieldUnits2 (read-only)
FieldValid (read-only)
First (read-only)
FitHeight
FitWidth
FlyAway (read-only)
Focus (read-only)
Font.Color
Font.Script (8)
Font.Size
Font.Style
Font.Typeface
Format.DateFormat
Format.LogicalFormat
Format.NumberFormat
Format.TimeFormat
Format.TimeStampFormat
Frame.Color
Frame.Style
Frame.Thickness
FullName (read-only)
FullSize (read-only)

HorizontalScrollBar
IndexField (read-only)
Inserting (read-only)
KeyField (read-only)
LabelText
LeftBorder (read-only) (5.0)
Locked (read-only)
LookupTable (read-only)
LookupType (read-only)
Magnification
Manager (read-only)
MarkerPos
Maximum (read-only)
Minimum (read-only)
Name
Next (read-only)
NextTabStop (5.0)
NoEcho
NRecords (read-only)
OverStrike
Owner (read-only)
Pattern.Color
Pattern.Style
Picture (enhanced for 5.0)
Position
Prev (read-only)
RasterOperation
Readonly
RecNo (read-only)
Refresh (read-only)
Required (read-only)
RightBorder (read-only) (5.0)
RowNo (read-only)
Scroll
Select (4.5)
SelectedText
SeqNo (read-only)
Size
SpecialField (5.0)
SummaryModifier (5.0)
TableName
TabStop
Text
TopBorder (read-only) (5.0)
TopLine
Touched (read-only)
Translucent
Value
VerticalScrollBar
Visible
WideScrollBar (5.0)
WordWrap

Form properties

Arrived (read-only)
AutoAppend (4.5)
BlankRecord (read-only)
Border
Caption
Class (read-only)
ContainerName (read-only)
ControlMenu
Deleted (read-only)
DesignModified (4.5)
DesktopForm
DialogForm
DialogFrame
Editing (read-only) (4.5)
Enabled (7)
FieldView (read-only)
First (read-only)
FlyAway (read-only)
Focus (read-only)
FrameObjects (5.0)
FullName (read-only)
FullSize (read-only)
GridValue (5.0)
GroupObjects (5.0)
HorizontalScrollBar
HTMLAction (8)
HTMLFormParams (8)
HTMLMethod (8)
Inserting (read-only)
Locked (read-only)
Manager (read-only)
MaximizeButton
MemoView (read-only)
MinimizeButton
Modal
MouseActivate
Name
Next (read-only)
NRecords (read-only)
PageTiling (5.0)
PersistView (read-only)
Position
Prev (read-only)
PrinterDocument (5.0)
RecNo (read-only)
Refresh (read-only)
Scroll
SeeMouseMove (5.0)
SeqNo (read-only)
ShowGrid (5.0)
Size (read-only)
SizeToFit
SnapToGrid (5.0)

StandardMenu
StandardToolbar (5.0)
TableName (read-only)
ThickFrame
Title
Touched (read-only)
VerticalScrollBar

Chart properties

Arrived (read-only)
Background.Color
Background.Pattern.Color
Background.Pattern.Style
BackWall.Color
BackWall.Pattern.Color
BackWall.Pattern.Style
BaseFloor.Color
BaseFloor.Pattern.Color
BaseFloor.Pattern.Style
BindType
BottomBorder (read-only) (5.0)
Class (read-only)
Color
ContainerName (read-only)
CurrentSeries
CurrentSlice
Design.ContainObjects
Design.PinHorizontal
Design.PinVertical
Design.Selectable (4.5)
Enabled (7)
First (read-only)
Focus (read-only)
Frame.Color
Frame.Style
Frame.Thickness
FullName (read-only)
FullSize (read-only)
GraphType
Label.Font.Color
Label.Font.Script (8)
Label.Font.Size
Label.Font.Style
Label.Font.Typeface
Label.LabelFormat
Label.LabelLocation
Label.NumberFormat
LeftBorder (read-only) (5.0)
LeftWall.Color
LeftWall.Pattern.Color
LeftWall.Pattern.Style
LegendBox.Color
LegendBox.Font.Color
LegendBox.Font.Script (8)
LegendBox.Font.Size
LegendBox.Font.Style
LegendBox.Font.Typeface
LegendBox.LegendPos
LegendBox.Pattern.Color
LegendBox.Pattern.Style
Manager (read-only)
MaxGroups

MaxXValues
MinXValues
Name
Next (read-only)
NextTabStop (5.0)
Options.Elevation
Options.Rotation
Options.ShowAxes
Options.ShowGrid
Options.ShowLabels
Options.ShowLegend
Options.ShowTitle
Owner (read-only)
Pattern.Color
Pattern.Style
Position
Prev (read-only)
RightBorder (read-only) (5.0)
Scroll
Select (4.5)
Series.Color
Series.Graph_Title.Font.Color
Series.Graph_Title.Font.Script (8)
Series.Graph_Title.Font.Size
Series.Graph_Title.Font.Style
Series.Graph_Title.Font.Typeface
Series.Graph_Title.Text
Series.Graph_Title.UseDefault
Series.Line.Color
Series.Line.LineStyle
Series.Line.Thickness
Series.Marker.Size (5.0)
Series.Marker.Style (5.0)
Series.Pattern.Color
Series.Pattern.Style
Series.TypeOverride
SeriesName (5.0)
Size
Slice.Color
Slice.Explode
Slice.Pattern.Color
Slice.Pattern.Style
TableName
TabStop
TitleBox.Color
TitleBox.Graph_Title.Font.Color
TitleBox.Graph_Title.Font.Script (8)
TitleBox.Graph_Title.Font.Size
TitleBox.Graph_Title.Font.Style
TitleBox.Graph_Title.Font.Typeface
TitleBox.Graph_Title.Text
TitleBox.Graph_Title.UseDefault
TitleBox.Pattern.Color
TitleBox.Pattern.Style
TitleBox.Subtitle.Font.Color
TitleBox.Subtitle.Font.Script (8)

TitleBox.Subtitle.Font.Size
TitleBox.Subtitle.Font.Style
TitleBox.Subtitle.Font.Typeface
TitleBox.Subtitle.Text
TitleBox.Subtitle.UseDefault
TitleBoxName (5.0)
TopBorder (read-only) (5.0)
Touched (read-only)
Translucent
Visible
XAxisName (5.0)
XAxis.Graph_Title.Font.Color
XAxis.Graph_Title.Font.Script (8)
XAxis.Graph_Title.Font.Size
XAxis.Graph_Title.Font.Style
XAxis.Graph_Title.Font.Typeface
XAxis.Graph_Title.Text
XAxis.Graph_Title.UseDefault
XAxis.Scale.AutoScale
XAxis.Scale.HighValue
XAxis.Scale.Increment
XAxis.Scale.Logarithmic
XAxis.Scale.LowValue
XAxis.Ticks.Alternate
XAxis.Ticks.DateFormat
XAxis.Ticks.Font.Color
XAxis.Ticks.Font.Script (8)
XAxis.Ticks.Font.Size
XAxis.Ticks.Font.Style
XAxis.Ticks.Font.Typeface
XAxis.Ticks.NumberFormat
XAxis.Ticks.TimeFormat (5.0)
XAxis.Ticks.TimeStampFormat (5.0)
YAxisName (5.0)
YAxis.Graph_Title.Font.Color
YAxis.Graph_Title.Font.Script (8)
YAxis.Graph_Title.Font.Size
YAxis.Graph_Title.Font.Style
YAxis.Graph_Title.Font.Typeface
YAxis.Graph_Title.Text (5.0)
YAxis.Graph_Title.UseDefault
YAxis.Scale.AutoScale
YAxis.Scale.HighValue
YAxis.Scale.Increment
YAxis.Scale.Logarithmic
YAxis.Scale.LowValue
YAxis.Ticks.Alternate
YAxis.Ticks.DateFormat
YAxis.Ticks.Font.Color
YAxis.Ticks.Font.Script (8)
YAxis.Ticks.Font.Size
YAxis.Ticks.Font.Style
YAxis.Ticks.Font.Typeface
YAxis.Ticks.NumberFormat
YAxis.Ticks.TimeFormat (5.0)
YAxis.Ticks.TimeStampFormat (5.0)

ZAxisName (5.0)
ZAxis.Graph_Title.Font.Color
ZAxis.Graph_Title.Font.Script (8)
ZAxis.Graph_Title.Font.Size
ZAxis.Graph_Title.Font.Style
ZAxis.Graph_Title.Font.Typeface
ZAxis.Graph_Title.Text (5.0)
ZAxis.Graph_Title.UseDefault
ZAxis.Scale.AutoScale
ZAxis.Scale.HighValue
ZAxis.Scale.Increment
ZAxis.Scale.Logarithmic
ZAxis.Scale.LowValue
ZAxis.Ticks.Alternate
ZAxis.Ticks.DateFormat
ZAxis.Ticks.Font.Color
ZAxis.Ticks.Font.Script (8)
ZAxis.Ticks.Font.Size
ZAxis.Ticks.Font.Style
ZAxis.Ticks.Font.Typeface
ZAxis.Ticks.NumberFormat (5.0)
ZAxis.Ticks.TimeFormat (5.0)
ZAxis.Ticks.TimeStampFormat (5.0)

Group properties

Arrived (read-only)
BottomBorder (read-only) (5.0)
Breakable
Class (read-only)
ContainerName (read-only)
Design.PinHorizontal
Design.PinVertical
Design.Selectable (4.5)
Enabled (7)
First (read-only)
Focus (read-only)
FullName (read-only)
FullSize (read-only)
LeftBorder (read-only) (5.0)
Manager (read-only)
Name
Next (read-only)
Owner (read-only)
Position
Prev (read-only)
RightBorder (read-only) (5.0)
Select (4.5)
Size
TopBorder (read-only) (5.0)
Visible

Header properties

Arrived (read-only)
BottomBorder (read-only) (5.0)
Class (read-only)
Color
ContainerName (read-only)
Design.ContainObjects
Design.PinHorizontal
Design.PinVertical
Design.Selectable (4.5)
Enabled (7)
First (read-only)
Focus (read-only)
FullName (read-only)
FullSize (read-only)
Invisible
LeftBorder (read-only) (5.0)
Manager (read-only)
Name
Next (read-only)
NextTabStop (5.0)
Owner (read-only)
Position
Prev (read-only)
RightBorder (read-only) (5.0)
Scroll
Select (4.5)
Size
TopBorder (read-only) (5.0)
Translucent
Visible

Line properties

Arrived (read-only)
BottomBorder (read-only) (5.0)
Breakable
Class (read-only)
Color
ContainerName (read-only)
Design.PinHorizontal
Design.PinVertical
Design.Selectable (4.5)
Enabled (7)
End
First (read-only)
Focus (read-only)
FullName (read-only)
FullSize (read-only)
Invisible
LeftBorder (read-only) (5.0)
LineEnds
LineStyle
LineType
Manager (read-only)
Name
Next (read-only)
Owner (read-only)
Position
Prev (read-only)
RightBorder (read-only) (5.0)
Select (4.5)
Size
Start
Thickness
TopBorder (read-only) (5.0)
Visible

List properties

Arrived (read-only)
BottomBorder (read-only) (5.0)
Class (read-only)
ContainerName (read-only)
DataSource
Design.PinHorizontal
Design.PinVertical
Design.Selectable (4.5)
Enabled (7)
First (read-only)
FitHeight
FitWidth
Focus (read-only)
FullName (read-only)
FullSize (read-only)
LeftBorder (read-only) (5.0)
List.Count
List.Selection
List.Value
Manager (read-only)
Name
Next (read-only)
Owner (read-only)
Position
Prev (read-only)
RightBorder (read-only) (5.0)
Scroll
Select (4.5)
Size
TopBorder (read-only) (5.0)
Visible
WideScrollBar (5.0)

Multi-record properties

Arrived (read-only)
AutoAppend (4.5)
BlankRecord (read-only)
BottomBorder (read-only) (5.0)
Breakable
ByRows (5.0)
Class (read-only)
Color
Columnar (4.5)
ContainerName (read-only)
CurrentColumn (5.0)
CurrentRow (5.0)
Deleted (read-only)
Editing (4.5)
Enabled (7)
First (read-only)
FirstRow (read-only) (5.0)
FitHeight
FitWidth
FlyAway (read-only)
Focus (read-only)
Frame.Color
Frame.Style
Frame.Thickness
FullName (read-only)
FullSize (read-only)
Inserting (read-only)
LeftBorder (read-only) (5.0)
Locked (read-only)
Manager (read-only)
Name
NCols
Next (read-only)
NRecords (read-only)
NRows
Owner (read-only)
Pattern.Color
Pattern.Style
Position
Prev (read-only)
Readonly (read-only)
RecNo (read-only)
Refresh (read-only)
RightBorder (read-only) (5.0)
RowNo (read-only)
Scroll
Select (4.5)
SeqNo (read-only)
Size
TableName
TopBorder (read-only) (5.0)
Touched
Translucent

VerticalScrollBar
Visible
WideScrollBar (5.0)
Xseparation (5.0)
Yseparation (5.0)

Notebook properties

Arrived (read-only)
BottomBorder (read-only)
Class (read-only)
ContainerName (read-only)
CurrentPage
Design.ContainObjects
Design.PinHorizontal
Design.PinVertical
Design.Selectable
Design.SizeToFit
Enabled
First (read-only)
Focus (read-only)
FullName (read-only)
FullSize (read-only)
HorizontalScrollBar
LeftBorder (read-only)
Manager (read-only)
Name
Next (read-only)
NextTabStop
NumberPages
Owner (read-only)
Position
Prev (read-only)
RightBorder (read-only)
Scroll
Select
Size
SquareTabs
TabHeight
TabsAcross
TabsOnTop
TopBorder (read-only)
Visible

Notebook page properties

InactiveColor
Enabled (7)

OLE properties

Arrived (read-only)
BottomBorder (read-only) (5.0)
Class (read-only)
ContainerName (read-only)
Design.ContainObjects
Design.PinHorizontal
Design.PinVertical
Design.Selectable (4.5)
Design.SizeToFit
Enabled (7)
First (read-only)
Focus (read-only)
Font.Script (8)
Frame.Color
Frame.Style
Frame.Thickness
FullName (read-only)
FullSize (read-only)
HorizontalScrollBar
LeftBorder (read-only) (5.0)
Magnification
Manager (read-only)
Name
Next (read-only)
NextTabStop (5.0)
Owner (read-only)
Position
Prev (read-only)
RightBorder (read-only) (5.0)
Scroll
Select (4.5)
Size
TopBorder (read-only) (5.0)
Value
VerticalScrollBar
Visible
WideScrollBar (5.0)

Page properties

Arrived (read-only)
BottomBorder (read-only) (5.0)
Class (read-only)
Color
ContainerName (read-only)
Enabled (7)
First (read-only)
Focus (read-only)
FullName (read-only)
FullSize (read-only)
LeftBorder (read-only) (5.0)
Manager (read-only)
Name
Next (read-only)
Owner (read-only)
Pattern.Color
Pattern.Style
Position (read-only)
PositionalOrder (5.0)
Prev (read-only)
RightBorder (read-only) (5.0)
Scroll
Select (4.5)
Size
TopBorder (read-only) (5.0)
Translucent
Visible

Record properties

Arrived (read-only)
AutoAppend (4.5)
BlankRecord (read-only)
BottomBorder (read-only) (5.0)
Breakable
Class (read-only)
Color
ContainerName (read-only)
Deleted (read-only)
DeleteWhenEmpty (5.0)
Design.ContainObjects
Design.PinHorizontal
Design.PinVertical
Design.Selectable (4.5)
Design.SizeToFit
Editing (read-only) (4.5)
Enabled (7)
First (read-only)
FitHeight
FitWidth
FlyAway (read-only)
Focus (read-only)
Frame.Color
Frame.Style
Frame.Thickness
FullName (read-only)
FullSize (read-only)
Inserting (read-only)
LeftBorder (read-only) (5.0)
Locked (read-only)
Manager (read-only)
Name
Next (read-only)
NRecords (read-only)
Owner (read-only)
Pattern.Color
Pattern.Style
Position
Prev (read-only)
Readonly
RecNo (read-only)
Refresh (read-only)
RightBorder (read-only) (5.0)
RowNo (read-only)
Scroll
Select (4.5)
SeqNo (read-only)
Shrinkable
Size
TableName (read-only)
TopBorder (read-only) (5.0)
Touched
Translucent

Visible

TableFrame properties

Arrived (read-only)
AttachedHeader (read-only) (5.0)
AutoAppend (4.5)
BlankRecord (read-only)
BottomBorder (read-only) (5.0)
Breakable
Class (read-only)
Color
ColumnPosition (5.0)
ColumnWidth (5.0)
ContainerName (read-only)
CurrentColumn (5.0)
CurrentRow (5.0)
DeleteColumn (5.0)
Deleted (read-only)
DeleteWhenEmpty (5.0)
Design.PinHorizontal
Design.PinVertical
Design.Selectable (4.5)
Design.SizeToFit
Editing (read-only) (4.5)
Enabled (7)
First (read-only)
FirstRow (read-only) (5.0)
FitHeight
FitWidth
FlyAway (read-only)
Focus (read-only)
FullName (read-only)
FullSize (read-only)
Grid.Color
Grid.GridStyle
Grid.RecordDivider
Header (read-only) (5.0)
HorizontalScrollBar
IncludeAllData (7)
InsertColumn (5.0)
Inserting (read-only)
LeftBorder (read-only) (5.0)
Locked (read-only)
Manager (read-only)
Name
NCols
Next (read-only)
NextTabStop (5.0)
NRecords (read-only)
NRows
Owner (read-only)
Pattern.Color
Pattern.Style
PersistView (read-only)
Position
Prev (read-only)

Readonly (read-only)
RecNo (read-only)
Refresh (read-only)
RepeatHeader (5.0)
RightBorder (read-only) (5.0)
RowNo (read-only)
Scroll
Select (4.5)
SeqNo (read-only)
ShowAllColumns (5.0)
Size
TableName
TopBorder (read-only) (5.0)
Touched
Translucent
VerticalScrollBar
Visible
WideScrollBar (5.0)

TableView properties

Arrived (read-only)
BlankRecord (read-only)
BottomBorder (read-only) (5.0)
Class (read-only)
Color
ContainerName (read-only)
CurrentRecordMarker.Color
CurrentRecordMarker.LineStyle
CurrentRecordMarker.Show
Deleted (read-only)
Editing (read-only) (4.5)
Enabled (7)
FieldName (read-only)
FieldNo (read-only)
FieldView (read-only)
Focus (read-only)
FullName (read-only)
FullSize (read-only)
GridLines.Color
GridLines.ColumnLines
GridLines.HeadingLines
GridLines.LineStyle
GridLines.QueryLook (5.0)
GridLines.RowLines
GridLines.Spacing
HeadingHeight (5.0)
Inserting (read-only)
LeftBorder (read-only) (5.0)
Locked (read-only)
Manager (read-only)
MemoView (read-only)
Name (read-only)
NCols (read-only)
NRecords (read-only)
NRows (read-only)
Owner (read-only)
PersistView (read-only)
Position
Readonly (read-only)
RecNo (read-only)
Refresh (read-only)
RightBorder (read-only) (5.0)
RowHeight (5.0)
RowNo
Scroll
Select (4.5)
SeqNo (read-only)
Size
TableName (read-only)
TopBorder (read-only) (5.0)
Touched (read-only)

Text properties

Alignment
Arrived (read-only)
AvgCharSize (read-only) (5.0)
BottomBorder (read-only) (5.0)
Breakable
Class (read-only)
Color
ContainerName (read-only)
CursorColumn
CursorLine
CursorPos
Design.ContainObjects
Design.PinHorizontal
Design.PinVertical
Design.Selectable (4.5)
Design.SizeToFit
DesignSizing
Enabled (7)
FieldSqueeze
First (read-only)
FitHeight
FitWidth
Focus (read-only)
Font.Color
Font.Script (8)
Font.Size
Font.Style
Font.Typeface
Frame.Color
Frame.Style
Frame.Thickness
FullName (read-only)
FullSize (read-only)
HorizontalScrollBar
InsertField (5.0)
LeftBorder (read-only) (5.0)
LineSpacing
LineSqueeze
Manager (read-only)
MarkerPos
Name
Next (read-only)
Orphan/Widow
OverStrike
Owner (read-only)
Pattern.Color
Pattern.Style
Position
Prev (read-only)
RightBorder (read-only) (5.0)
Scroll
Select (4.5)
SelectedText

Size
Text
TopBorder (read-only) (5.0)
TopLine
Translucent
Value
VerticalScrollBar
Visible
WideScrollBar (5.0)
WordWrap

TVData properties

Alignment
Arrived (read-only)
BlankRecord (read-only)
BottomBorder (read-only) (5.0)
Class (read-only)
Color
CompleteDisplay
ContainerName (read-only)
Default (read-only)
Deleted (read-only)
Design.ContainObjects
Design.PinHorizontal
Design.PinVertical
Design.Selectable (4.5)
Design.SizeToFit
Editing (read-only) (4.5)
FieldName (read-only)
FieldNo (read-only)
FieldRights (read-only)
FieldSize (read-only)
FieldType (read-only)
FieldUnits2 (read-only)
First (read-only)
Focus (read-only)
Font.Color
Font.Script (8)
Font.Size
Font.Style
Font.Typeface
Format.DateFormat
Format.LogicalFormat
Format.NumberFormat
Format.TimeFormat
Format.TimeStampFormat
FullName (read-only)
FullSize (read-only)
IndexField (read-only)
Inserting
KeyField (read-only)
LeftBorder (read-only) (5.0)
Locked (read-only)
LookupTable (read-only)
LookupType (read-only)
Magnification
Manager (read-only)
Maximum (read-only)
Minimum (read-only)
Name
NextTabStop (5.0)
NRecords (read-only)
Owner (read-only)
Picture (read-only)
Position

RecNo (read-only)
Refresh (read-only)
Required (read-only)
RightBorder (read-only) (5.0)
RowNo (read-only)
Scroll
Select (4.5)
SeqNo (read-only)
Size
TableName (read-only)
TopBorder (read-only) (5.0)
Touched (read-only)
Value
Width (5.0)

TVHeading properties

Alignment
Arrived (read-only)
BottomBorder (read-only) (5.0)
Class (read-only)
Color
ContainerName (read-only)
Design.ContainObjects
Design.PinHorizontal
Design.PinVertical
Design.Selectable (4.5)
Design.SizeToFit
Focus (read-only)
Font.Color
Font.Script (8)
Font.Size
Font.Style
Font.Typeface
FullName (read-only)
FullSize (read-only)
LeftBorder (read-only) (5.0)
Manager (read-only)
Name
Owner (read-only)
Position
RightBorder (read-only) (5.0)
Scroll
Select (4.5)
Size
TopBorder (read-only) (5.0)

List of built-in event methods

The following table lists built-in event methods for internal and external events and special built-in event methods. Click one of the following methods for more information.

Internal	External	Special
arrive	action	pushButton
canArrive	error	changeValue
canDepart	init	newValue
close	keyChar	
depart	keyPhysical	
mouseEnter	menuAction	
mouseExit	mouseClick	
open	mouseDouble	
removeFocus	mouseDown	
setFocus	mouseMove	
timer	mouseRightDouble	
	mouseRightDown	
	mouseRightUp	
	mouseUp	
	status	

{button ,AL(`BUILTIN;INTRO_COMPONENTS_METH_EV;',0,"Defaultoverview",)} **Related Topics**

Print related ObjectPAL methods and examples

About built-in event methods

Every object in a form (and the form itself) has built-in event methods for handling events. Built-in event methods have the same names as the events that trigger them. For example, changing a value triggers an object's built-in **changeValue** method, pressing the mouse button triggers the built-in **mouseDown** method, and releasing the mouse button triggers the built-in **mouseUp** method. The behavior of an object is simply the combined effects of its built-in event methods. ObjectPAL follows a specific sequence of execution, moving from one built-in event method to the next.

There are three kinds of built-in event methods in ObjectPAL:

- built-in event methods for internal events
- built-in event methods for external events
- special built-in event methods

Built-in event methods for internal events

Internal events are generated internally by ObjectPAL. Like all events, internal events go to the form first, which dispatches them to the target object. Internal events do not bubble up through the containership hierarchy.

Built-in event methods for external events

External events are typically generated by user actions, although they can also be generated by ObjectPAL statements. Processing for all external events begins with the form, which acts as a dispatcher. Any external event that can not be handled by an object bubbles up the containership hierarchy.

Unless otherwise noted, the default behavior for most objects and built-in event methods is to pass the event up the containership hierarchy .

Special built-in event methods

Special built-in event methods are additional methods built into a few specific objects.

{button ,AL(`BUILTIN;INTRO_COMPONENTS_METH_EV;',0,"Defaultoverview",)} Related Topics

Editing a built-in event method

Every Corel Paradox object comes with built-in event methods (e.g., **open**, **close**, and **mouseUp**) for each event it can respond to. The built-in code for these methods specify an object's default behavior in response to a given event. You can edit this code using the ObjectPAL Editor.

To edit a built-in event method

1. Open the Object Explorer.
2. Click the Events tab.
3. Double-click the method where you want to attach the code.
4. In the Editor window, make any changes.

You can type the text for a method directly in the ObjectPAL Editor, or use the Clipboard to copy, cut, and paste methods and parts of methods from other objects.

Now your code executes whenever this event method you've chosen is called for the object specified.

If you've attached code to the built-in event method, the built-in code executes *after* your code.

The built-in code is implicit and executes automatically. But, you can change the default behavior; for example, by calling the built-in code *before* your code, or blocking it from executing. You should first understand the default behavior for each built-in event method.

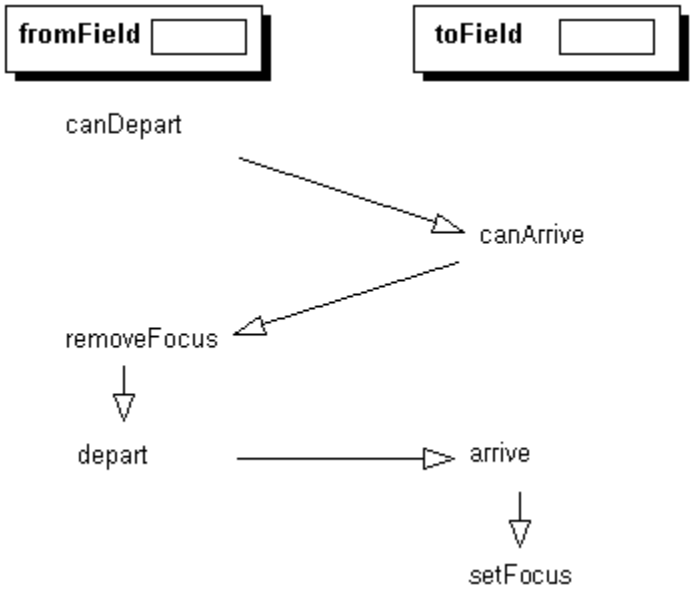
Notes

- If you attach code to an object's built-in **open** method, you should call **doDefault** before calling any other method or procedure. The call to **doDefault** executes the built-in code so you can be sure the object is completely opened and initialized. Calling **disableDefault** stops the built-in code from executing.
- To edit more than one method, select them by pressing SHIFT and clicking, and then press ENTER. An Editor window opens for each event selected.

{button ,AL(`BUILTIN;OPAL_BLANG_BLDISABLE:INTRO_COMPONENTS_METH;INTRO_COMPONENTS_METH_EV;OPAL_BLANG_BLDODEFAULT;OPAL_BLANG_BLENABLE;',0,"Defaultoverview",)} Related Topics

Sequence of execution

The following figure shows the sequence in which built-in event methods execute when you move from one field object to another (for example, by pressing TAB). In this example, the field object you're moving from is *fromField*, and the one you're moving to is *toField*.



{button ,AL(`BUILTIN;INTRO_COMPONENTS_METH_EV;',0,"Defaultoverview",)} **Related Topics**

init method

The **init** built-in event method allows you to perform any initialization operations before a form's **open** method is executed. **init** is called once when a form window is opened to view data. **init** is a built-in method only to the form object; no other UIObject has a built-in **init** method.

In previous versions of Corel Paradox, initialization code was placed in the **open** method of the form. The **init** method executes at the same point that the call to **open** previously occurred. However, **open** is now called as the default behavior once the **init** method has executed. While the **init** method is the preferred location for form-level initialization code, existing forms with initialization code placed in the PreFilter block of the **open** method will continue to behave as before. Unlike all other form-level methods, **init** does not have a PreFilter clause.

Both the **init** and **open** methods can contain initialization code for your application. Because of the PreFilter clause, however, the **open** method is called once for every object on the form. Depending on the number of objects on the form, this may cause a form to load slowly. **init** allows for pre-processing or initialization code to be executed only once—at the form level.

The **init** method can be used to stop the form from executing (as the **open** method can) by setting the error code to any non-zero value as follows:

```
eventInfo.setErrorCode(1)      ; any nonzero error code will work
```

The **init** method calls the **open** method, so any error code returned from the **open** method bubbles up to **init**. Any forms that return error codes in their **open** continue to stop the form from executing.

Examples

To setup tables before the **open** method opens them, write the following code :

```
method init(var eventInfo Event)
createMyTables()     ; createMyTables is a custom method
                     ; implicit doDefault here calls the open method
endMethod
```

To have the system open the tables before processing them, write this code :

```
method init(var eventInfo Event)
doDefault                 ; open method is executed here
tc.attach(tableFrame)     ; add any special processing code here.
                          ; tc is defined in the Var window.
endMethod
```

{button ,AL(`BUILTIN;INTRO_COMPONENTS_METH_EV;INTRO_COMPONENTS_METH;,')} Related Topics

open

open is called once for every object on the form, starting from the form and working down each container in turn. For every object, the default code for the **open** method calls the **open** method for each of its child objects (that is, the objects one level below it in the containership hierarchy). In other words, by default, the form's **open** method calls the **open** method for each page in the form, and each page's **open** method calls the **open** method for each object on the page, and so on.

 **Note**

- The form's **open** method opens all tables in the form's data model *before* any other objects are opened. If aliases are required to open any of the tables, specify them before executing the default code for **open**.

An error from any object prevents a form from opening. Also, explicitly setting an error code in an **open** method's event packet prevents a form from opening.

As a general rule, if you attach code to an object's built-in **open** method, you should call **doDefault** before calling any other method or procedure. The call to **doDefault** executes the built-in code so you can be sure the object is completely opened and initialized.

 **Note**

- Corel Paradox compiles out of date forms (forms created in earlier versions) before opening them. In this case, tables in the form's data model are opened before any ObjectPAL code executes.

{button ,AL(`BUILTIN;INTRO_COMPONENTS_METH_EV;INTRO_COMPONENTS_METH;,')} Related Topics

close

close is called once for every object on a form being closed. By default, a form's **close** method closes all tables attached to the form.

{button ,AL(`BUILTIN;INTRO_COMPONENTS_METH_EV;INTRO_COMPONENTS_METH;,')} Related Topics

canArrive

canArrive is called when moving to an object. It finds out whether the object (usually a field object) can be made active. Corel Paradox calls this method by working through the object's containers, triggering **canArrive** for each object until it reaches the object itself. At any level of the containership, an error denies permission, blocking the move.

By default, Corel Paradox blocks **canArrive** for objects that are not tab stops, and for a crosstab object if the target is not a cell.

{button ,AL(`BUILTIN;INTRO_COMPONENTS_METH_EV;INTRO_COMPONENTS_METH;,')} Related Topics

arrive

arrive is called only after the target and its containers have allowed a **canArrive**. As with **canArrive**, Corel Paradox calls **arrive** on the target object's containers, finishing at the target. Pages, table frames, and multi-record objects all move to the first tab stop object they contain if they are the final destination of the move.

A successful **arrive** on a record or a field object makes the record or object active (and opens an Editor window for the field object, if appropriate).

arrive can have further effects on a field object, depending on the object's display type. If the object is a drop-down edit list, focus moves to the list. If it's a radio button, focus moves to the first button.

{button ,AL(`BUILTIN;INTRO_COMPONENTS_METH_EV;INTRO_COMPONENTS_METH;,')} Related Topics

setFocus

setFocus is called after a successful **arrive** or when focus is returned to the form after moving to another window. This method is called for each of the active object's containers, starting with the outermost container, before it is called for the active object itself.

On an edit field, the default code for **setFocus** highlights the active selection and makes the cursor blink. At this time, also, the object's Focus property is set to True, and the form displays a status message reporting the number of the active record and the total number of records.

When a button gets focus, a rectangle displays around the label.

{button ,AL(`BUILTIN;INTRO_COMPONENTS_METH_EV;INTRO_COMPONENTS_METH;,')} Related Topics

canDepart

canDepart is called when you try to move the focus off any object. This is the place to attach code to block a departure; any error blocks the move. Field objects try to post their contents (triggering **changeValue**), and record objects try to commit the active record if changes have been made. If the record is locked, the form calls **action(DataUnlockRecord)** or **action(DataPostRecord)**.

Switching to another window does not move off an object, it only changes focus. Use **setFocus** and **removeFocus** to respond to focus events.

{button ,AL(`BUILTIN;INTRO_COMPONENTS_METH_EV;INTRO_COMPONENTS_METH;,')} Related Topics

removeFocus

removeFocus removes the flashing cursor and highlight (if appropriate) from a field object, and removes the rectangle from a button. The object's Focus property is set to False.

The removeFocus method is called for the active object and all its containers, starting with the active object, when the user activates some other window or moves to some other object.

{button ,AL(`BUILTIN;INTRO_COMPONENTS_METH_EV;INTRO_COMPONENTS_METH;,')} Related Topics

depart

depart is called after all containers of the active object have been granted permission to leave the field via **canDepart** and **removeFocus**. Use **canDepart** to block a move; **depart** is reserved for closing edit regions, and performing general cleanup.

{button ,AL(`BUILTIN;INTRO_COMPONENTS_METH_EV;INTRO_COMPONENTS_METH;,')} Related Topics

mouseEnter

mouseEnter is called whenever the pointer crosses into an object. It is called only on the transition into the object, not on every move across the object.

By default, field objects set the pointer to the I-beam; form, page, and button objects set the pointer to an arrow.

If a button was the last object to be clicked, and the mouse button is still pressed, the button's value toggles between True and False.

{button ,AL(`BUILTIN;INTRO_COMPONENTS_METH_EV;INTRO_COMPONENTS_METH;,')} Related Topics

mouseExit

mouseExit is called when the pointer leaves an object. An object that changes the shape of the pointer on **mouseEnter** sets it to an arrow in **mouseExit**.

If a button was the last object to be clicked, and the mouse button is still pressed, the button's value toggles between True and False.

{button ,AL(`BUILTIN;INTRO_COMPONENTS_METH_EV;INTRO_COMPONENTS_METH;,')} Related Topics

timer

timer is called whenever a timer interval elapses. Use the UIObject method **setTimer** to set timer intervals.

{button ,AL(`BUILTIN;INTRO_COMPONENTS_METH_EV;INTRO_COMPONENTS_METH;,')} Related Topics

mouseDown

mouseDown is called when the logical left mouse button is pressed. The event packet for this method contains the mouse coordinates in twips, relative to the target object.

An active field object enters Field View, positions the cursor, and begins a drag-selection.

When the form handles a **mouseDown**, it calls **mouseExit** for all objects no longer under the mouse, and calls **mouseEnter** for all objects now under the mouse. The form then dispatches the **mouseDown** to the object the mouse was pointing at.

This method toggles a button's value between True and False.

{button ,AL(`BUILTIN;INTRO_COMPONENTS_METH_EV;INTRO_COMPONENTS_METH;,')} Related Topics

mouseUp

mouseUp is called when the left mouse button is released. It's called for the last object to receive a **mouseDown** so the object always sees the **mouseDown**/**mouseUp** pair, even if the button is released outside the object.

An active field object ends the selection; a field object that is not active performs a **self.moveTo()**.

When the form handles a **mouseUp**, it calls **mouseExit** for all objects no longer under the mouse, and calls **mouseEnter** for all objects now under the mouse. The form then dispatches the **mouseUp** to the object that received the last click.

The **mouseUp** method toggles a button's value between True and False. If **mouseUp** is called and the pointer is inside a button, it triggers the button's **pushButton** method. For any other type of object, it triggers the object's built-in **mouseClick** method.

{button ,AL(`BUILTIN;INTRO_COMPONENTS_METH_EV;INTRO_COMPONENTS_METH;,')} Related Topics

mouseDouble

mouseDouble is called when the left mouse button is double-clicked. In Windows convention, a **mouseDown** and **mouseUp** are delivered first.

Field objects enter field view on a **mouseDouble**.

When the form handles a **mouseUp**, it calls **mouseExit** for all objects no longer under the mouse, and calls **mouseEnter** for all objects now under the mouse. The form then dispatches **mouseDouble** to the object that received the last click.

{button ,AL(`BUILTIN;INTRO_COMPONENTS_METH_EV;INTRO_COMPONENTS_METH;,')} Related Topics

mouseRightDown

mouseRightDown is called when the logical left mouse button is pressed. The event packet for this method contains the mouse coordinates in twips, relative to the target object.

An active field object enters Field View, positions the cursor and begins a drag-selection.

When the form handles a **mouseRightDown**, it calls **mouseExit** for all objects no longer under the mouse, and calls **mouseEnter** for all objects now under the mouse. The form then dispatches the **mouseRightDown** to the object the mouse was pointing at.

This method toggles a button's value between True and False.

{button ,AL(`BUILTIN;INTRO_COMPONENTS_METH_EV;INTRO_COMPONENTS_METH;,')} Related Topics

mouseRightUp

mouseRightUp is called when the left mouse button is released. It's called for the last object to receive a **mouseRightDown** so the object always sees the **mouseRightDown**/**mouseRightUp** pair, even if the button is released outside the object.

When the form handles a **mouseRightUp**, it calls **mouseExit** for all objects no longer under the mouse, and calls **mouseEnter** for all objects now under the mouse. The form then dispatches the **mouseRightUp** to the object that received the last click.

This method toggles a button's value between True and False. If **mouseRightUp** is called and the pointer is inside a button, it triggers the button's **pushButton** method.

In addition, the following field objects display a pop-up menu when they receive a **mouseRightUp**: formatted memo, graphic, OLE, and unbound (undefined).

{button ,AL(`BUILTIN;INTRO_COMPONENTS_METH_EV;INTRO_COMPONENTS_METH;,')} Related Topics

mouseRightDouble

mouseRightDouble is called when the left mouse button is double-clicked. In Windows convention, a **mouseRightDown** and **mouseRightUp** are delivered first.

When the form handles a **mouseRightUp**, it calls **mouseExit** for all objects no longer under the mouse, and calls **mouseEnter** for all objects now under the mouse. The form then dispatches **mouseRightDouble** to the object that received the last click.

{button ,AL(`BUILTIN;INTRO_COMPONENTS_METH_EV;INTRO_COMPONENTS_METH;,')} Related Topics

mouseClick

mouseClick is called when the logical left mouse button is pressed and released when the pointer is inside the boundaries of an object. **mouseClick** is not called if the user moves the mouse outside the object before releasing the mouse button.

mouseClick is generated from within the **mouseUp** method.

The mapping from **mouseUp** to **mouseClick** happens at the first container object which uses **mouseUp**. In other words, **mouseUp** in a box bubbles to its container. Only the field object, the button object, the list object, and the form intercept **mouseUp**; those are the only areas where the translation occurs. If you click on an object inside a button, that object's **mouseClick** will be called. If that object allows the default (bubbling), then the button will ultimately receive that **mouseClick**, triggering its own **pushButton** method. In this way, you can have code execute on objects you click inside the button, but still trigger the button's **pushButton** method. Setting the error code in **mouseUp** will inhibit the **mouseClick**, and setting the error code in **mouseClick** will inhibit a **pushButton**.

{button ,AL(`BUILTIN;INTRO_COMPONENTS_METH_EV;INTRO_COMPONENTS_METH;,')} Related Topics

mouseMove

mouseMove is called whenever the mouse moves within an object. The event packet for this method contains the coordinates of the pointer (in twips).

An active edit field checks the state of SHIFT. If SHIFT is down (physically or logically), the area being selected is extended. An active graphic field scrolls to show the entire object, if it is too large for it's current frame.

When you press and hold the mouse button inside an object, **mouseMove** is called until you release it, even when the pointer moves outside the object.

When the form handles a **mouseMove**, it calls **mouseExit** for all objects no longer under the mouse, and calls **mouseEnter** for all objects now under the mouse.

{button ,AL(`BUILTIN;INTRO_COMPONENTS_METH_EV;INTRO_COMPONENTS_METH;,')} Related Topics

keyPhysical

keyPhysical is called when a key is pressed and each time a key auto-repeats. This method goes to the form first, and the form dispatches it to the active object. Then, the active object's built-in code decides whether a keystroke represents an action or a character displayed in a field, and calls the appropriate **action** or **keyChar** method.

For example, suppose a field object within a table object is active and the user presses ENTER. The keystroke triggers **keyPhysical**, which interprets this action as a request and maps it to **action(FieldEnter)**, which in turn triggers the built-in **action** method. In contrast, when the user presses K, the keystroke triggers **keyPhysical**, which interprets it as a character and triggers the **keyChar** method.

The event packet for **keyPhysical** contains information from the Windows WM_KEYDOWN message and an optional WM_CHAR. Therefore, the event packet provides both the virtual key code and the ANSI character. This method is best for special character handling. For example, if you want to intercept the F9 key explicitly (rather than handle the eventual **action(DataToggleEdit)**), this is the method to use.

{button ,AL(`BUILTIN;INTRO_COMPONENTS_METH_EV;INTRO_COMPONENTS_METH;,')} Related Topics

keyChar

keyChar is called when a **keyPhysical** does not map to an action (see the **action** built-in event method). **keyChar** goes to the form first, and the form dispatches it to the active object.

When editing a field, the system locks the record before inserting the first character.

If a button receives a **keyChar** equal to pressing SPACEBAR (for example, **keyChar(VK_SPACE)**), it calls the button's **pushButton** method.

{button ,AL(`BUILTIN;INTRO_COMPONENTS_METH_EV;INTRO_COMPONENTS_METH;,')} Related Topics

menuAction

menuAction is called whenever the user chooses an item from a menu (or clicks a Toolbar button that executes a menu action). It goes to the form first, and the form dispatches it to the active object.

{button ,AL(`BUILTIN;INTRO_COMPONENTS_METH_EV;INTRO_COMPONENTS_METH;,')} Related Topics

error

error is called when an error occurs. By default, objects (except forms) pass errors to their containers. You can attach code to the default method to make an object handle an error, pass an error, or both.

{button ,AL(`BUILTIN;INTRO_COMPONENTS_METH_EV;INTRO_COMPONENTS_METH;,')} Related Topics

status

status is called before a message is displayed in one of the areas in the Status Bar. Among other things, you can attach code to the built-in **status** method to redirect messages to other areas or to change the text of the message.

{button ,AL(`BUILTIN;INTRO_COMPONENTS_METH_EV;INTRO_COMPONENTS_METH;,')} Related Topics

action

action is called when **keyPhysical** maps a keystroke to an action, when **menuAction** maps a menu command to an action, or when other methods want an action performed. **action** goes to the form first, and the form dispatches it to the target object. For example, by default, pressing F2 in a field triggers **action(EditToggleFieldView)** after its **keyPhysical** method executes, and clicking the Forward navigation button triggers **action(DataNextRecord)** after its **menuAction** method executes.

{button ,AL(`BUILTIN;INTRO_COMPONENTS_METH_EV;INTRO_COMPONENTS_METH;,')} Related Topics

pushButton

pushButton is defined only for button objects and fields displayed as list boxes, and is called when the user releases the mouse on a button. This method is not called directly by the form, but by the default **mouseUp** method for buttons. You can also call this method directly to accomplish the normal action associated with pressing a button object.

By default, buttons change their appearance when clicked. For example, a push button pushes in and pops out, check boxes check or uncheck, and a radio button pushes in or pops out. Focus moves to a button when the button is clicked (unless its Tab Stop property is set to False).

There are two ways to trigger a button's **pushButton** method using the keyboard when the button's Tab Stop property is set to True, and the button is the active object:

- Press SPACEBAR. The button keeps focus.
- Press ENTER. The focus moves to the next object in the tab order.

{button ,AL(`BUILTIN;INTRO_COMPONENTS_METH_EV;INTRO_COMPONENTS_METH;,')} Related Topics

changeValue

Defined only for field objects, **changeValue** asks for permission to change the value of a field. It is called before the value is stored, so you can check the value and do something with it (such as performing additional validity checks). **changeValue** is not called when someone changes a value across a network.

The following statement triggers Quant's **changeValue** method. Even if Quant is already 10, it triggers the statement immediately, without waiting for the method to finish executing.

Quant = 10

Using changeValue with field objects

The built-in code for **changeValue** commits the changes to the value. Until the built-in code executes, Corel Paradox uses the old, unchanged value. For example, suppose a field object has a value of 10, and you move to the value and enter a value of 23. Then, when you move off that field object, you trigger its **changeValue** method, to which the following code has been attached:

```
method changeValue(var eventInfo ValueEvent)
   msgInfo("before the change", self.value) ; displays 10
   doDefault
   msgInfo("after the change", self.value) ; displays 23
endMethod
```

When this method executes, the first dialog box displays the old value, 10, because the built-in code has not yet executed. Then, the call to **doDefault** executes the built-in code, which commits the changed value, and the second dialog box displays the changed value.

Within an object's **changeValue** method, you can use the ValueEvent type **newValue** method (this is different from the built-in **newValue** event method) to get the incoming value before the built-in code executes. For example, suppose that a field object has a value of 10, and you move to the value, enter a value of 23, and trigger its **changeValue** method, to which the following code has been attached:

```
method changeValue(var eventInfo ValueEvent)
   msgInfo("before the change", self.value) ; Displays 10
   msgInfo("the new (incoming) value",
           eventInfo.newValue())            ; Displays 23
   doDefault
   msgInfo("after the change", self.value)  ; Displays 23
endMethod
```

The first dialog box displays the old, unchanged value. The second dialog box calls **eventInfo.newValue** to display the new, incoming value, but that value has not yet been committed. The call to **doDefault** executes the built-in code, which commits the change, and the third dialog box displays the changed value.

You can block an attempted change to a value by calling **eventInfo.setErrorCode** and specifying a non-zero value. You can also test (or alter) the incoming value (e.g., to round up to the nearest dollar amount), using **eventInfo.setNewValue**. For example, the following code is attached to a field object's built-in **changeValue** method. It executes when the user changes the field's value and attempts to post (commit) the change. The code calls **eventInfo.newValue** to get the user's value before it is posted. If it is greater than 50, the call to **eventInfo.setErrorCode** prevents the user from posting the value or leaving the field.

```
method changeValue(var eventInfo ValueEvent)
   var
      atNewVal   AnyType
   endVar


   atNewVal = eventInfo.newValue()
   if atNewVal > 50 then
      eventInfo.setErrorCode(CanNotDepart)
      message("Enter a value less than 50.")
   endIf
endMethod
```

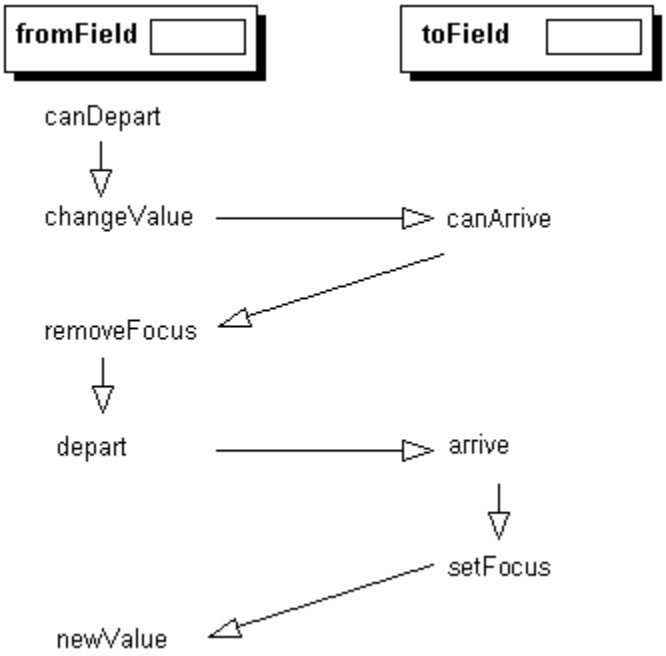
{button ,AL(`BUILTIN;INTRO_COMPONENTS_METH_EV;INTRO_COMPONENTS_METH;,')} Related Topics

newValue

Defined only for field objects, **newValue** is called to report that a field object has a new value. For example, moving to the next record in a table triggers **newValue**. When a field is displayed as radio buttons, **newValue** is called when you click a button. Note that typing into a field object does not trigger **newValue,** but it does set the Touched property to True. **changeValue** is not called until you try to move off the field object or otherwise try to commit changes. Also, a form's **open** method triggers **newValue** for each field object in the form.

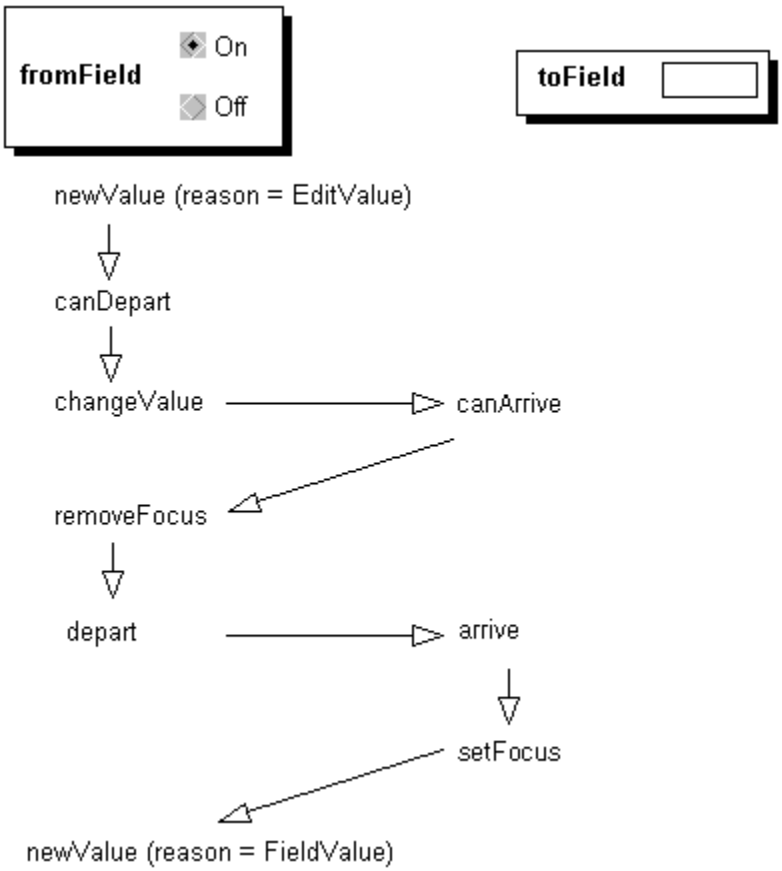
Labeled and unlabeled field objects

The following figure shows the sequence in which built-in event methods execute when you move from one field object (labeled or unlabeled) to another after editing a value. The field object you're moving from is named fromField; the one you're moving to is named toField.



Radio buttons and lists

The following figure shows the sequence in which built-in event methods execute when you move from one field object (radio buttons, lists, or check boxes) to another after editing a value. The sequence is the same as for regular fields, except for an additional **newValue** call when you choose a radio button or a list item. The table also gives the ValueReason constant for each **newValue**. The field object you're moving from is *fromField*; the one you're moving to is *toField*.



Note

- **newValue** is called when Corel Paradox needs to refresh the value of the field object (in this case, to update the display). Calls to **newValue** are not part of the **canDepart** sequence. However, **newValue** is not necessarily the last method to execute.

{button ,AL(`BUILTIN;INTRO_COMPONENTS_METH_EV;INTRO_COMPONENTS_METH;,')} Related Topics

ActionEvent type

ActionEvents are generated primarily by editing and navigating in a table. The ActionEvent type includes several derived methods from the Event type.

The only built-in event method that is triggered by an ActionEvent is **action**. Typically, when you work with ActionEvents, you'll also work with ObjectPAL action constants. For example, to prevent users from editing a table, you could do something like this:

```
; thisTableFrame::action
method action(var eventInfo ActionEvent)
; If the user tries to switch to Edit mode, display a dialog box
if eventInfo.id() = DataBeginEdit then  ; DataBeginEdit is a constant.
  msgStop("Stop", "You can't edit this table.")
  eventInfo.setErrorCode(UserError) ; UserError is a constant.
endif
endMethod
```

The action constants are grouped as follows:

- ActionDataCommands
- ActionEditCommands
- ActionFieldCommands
- ActionMoveCommands
- ActionSelectCommands

You can also use user-defined action constants.

The following table displays the methods for the ActionEvent type:

Methods for the ActionEvent type

Event	←	**ActionEvent**
errorCode		**actionClass**
getTarget		**id**
isFirstTime		**setId**
isPreFilter		
isTargetSelf		
reason		
setErrorCode		
setReason		

{button ,AL(`OPAL_ACTIONEVENT_USERDEFINEDCONSTANTS;',0,"Defaultoverview",)} **Related Topics**

Print related ObjectPAL methods and examples

User-defined action constants

You can define your own action constants, but you must keep them within a specific range. Because this range is subject to change in future versions of Corel Paradox, ObjectPAL provides the IdRanges constants UserAction and UserActionMax to represent the minimum and maximum values allowed.

For example, suppose that you want to define two action constants, ThisAction and ThatAction. In a Const window, define values for your custom constants as follows:

```
Const
   ThisAction = 1
   ThatAction = 2
EndConst
```

Then, to use one of these constants, add it to UserAction. For example,

```
method action(var eventInfo ActionEvent)


   if eventInfo.id() = UserAction  +  ThisAction then
      doSomething()
   endIf
endMethod
```

By adding UserAction to your own constant, you guarantee yourself a value above the minimum. To keep the value under the maximum, use the value of UserActionMax. One way to check the value is with a **message** statement:

```
message(UserActionMax)
```

In Corel Paradox, the difference between UserAction and UserActionMax is 2047. That means the largest value you can use for an action constant is UserAction + 2047.

{button ,AL(`opal_type_actionevent;;;;;',0,"Defaultoverview",)} **Related Topics**

actionClass method

Returns the class number of an ActionEvent.

Syntax

```
actionClass ( ) SmallInt
```

Description

actionClass returns an integer value representing an ActionEvent class. Use ActionClasses constants to find out which class the integer value represents.

Example

{button ,AL(`OPAL_TYPE_ACTIONEVENT;OPAL_METH_AEID;OPAL_METH_AESID;OPAL_METH_EVGTAR;',0,"Defaultoverview",)} Related Topics

actionClass example

The following example uses **actionClass** to prevent the user from making any changes to a field object. This code is attached to a field's built-in **action** method. See **id** for an example that traps for the user entering Edit mode.

```
; Site_Notes::action
method action(var eventInfo ActionEvent)
; check for any attempt to edit, and block it
if eventInfo.actionClass() = EditAction then
  ; allow user to start and end field view
  if NOT (eventInfo.id() = EditEnterFieldView) AND
     NOT (eventInfo.id() = EditToggleFieldView) AND
     NOT (eventInfo.id() = EditExitFieldView) then
     eventInfo.setErrorCode(UserError)
     beep()
     message("Sorry. Can't make changes to this field.")
  endif
endif
endMethod
```

id method

Returns the ID number of an ActionEvent.

Syntax

```
id ( ) SmallInt
```

Description

id returns the ID number of an ActionEvent. ObjectPAL defines constants for these ID numbers (for example, DataBeginEdit), so you don't have to remember numeric values.

The action constants are grouped as follows:

- ActionDataCommands
- ActionEditCommands
- ActionFieldCommands
- ActionMoveCommands
- ActionSelectCommands

You can also use user-defined action constants.

Example

{button ,AL(`OPAL_TYPE_ACTIONEVENT;OPAL_METH_AEACTIONCLASS;OPAL_METH_AESID;',0,"Defaultoverview",)} Related Topics

id example

The following example uses **id** to prevent the user from entering Edit mode on a form. This code is attached to a form's built-in **action** method:

```
; thisForm::action
method action(var eventInfo ActionEvent)
if eventInfo.isPreFilter() then
    ; code here executes for each object in form
else
    ; code here executes just for form itself
    if eventInfo.id() = DataBeginEdit then
      eventInfo.setErrorCode(UserError)     ; don't start Edit mode
      msgStop("Sorry", "View only - can't edit this form")
    endif
endif
endMethod
```

setId method

Specifies an ActionEvent.

Syntax

```
setId ( const actionId SmallInt )
```

Description

setId specifies the ActionEvent represented by the constant *actionId*. ObjectPAL provides constants (e.g., DataNextRecord) for ActionEvents so you don't have to remember numeric values.

The action constants grouped as follows:

- ActionDataCommands
- ActionEditCommands
- ActionFieldCommands
- ActionMoveCommands
- ActionSelectCommands

You can also use user-defined action constants.

Example

{button ,AL(`OPAL_TYPE_ACTIONEVENT;OPAL_METH_AEID;',0,"Defaultoverview",)} Related Topics

setId example

In the following example, the Toolbar record-movement buttons are remapped to move within a memo field. Assume that a form contains a multi-record object, *SITES*, bound to the *Sites* table. The following code is attached to the **action** method for the *Site_Notes* field object:

```
; Site_Notes::action
method action(var eventInfo ActionEvent)
var
  actID  SmallInt
endVar
; if Site Notes is in Field View, remap record-movement
; actions to move within the memo field
if self.Editing then
   actID = eventInfo.id()
   switch
      case actID = DataPriorRecord  : eventInfo.setId(MoveBeginLine)
      case actID = DataNextRecord   : eventInfo.setID(MoveEndLine)
      case actID = DataFastBackward : eventInfo.setID(MoveBegin)
      case actID = DataFastForward  : eventInfo.setID(MoveEnd)
      case actID = DataBegin        : eventInfo.setID(FieldBackward)
      case actID = DataEnd          : eventInfo.setID(FieldForward)
   endswitch
endif
endMethod
```

AnyType type

An AnyType variable can store any one of the data types listed in the following table.

Type	Description
AnyType	Any basic data type
Binary	Machine-readable data
Currency	Used to manipulate currency values
Date	Calendar data
DateTime	Calendar and clock data combined
Graphic	A bitmap image
Logical	True or False
LongInt	Used to represent large integer values
Memo	Holds a large amount of text
Number	Floating-point values
OLE	A link to another application
Point	Information about a location on the screen
SmallInt	Used to represent relatively small integer values
String	Letters
Time	Clock data

An AnyType variable can never be a complex type such as TCursor or TextStream. It inherits characteristics from the value assigned to it, behaving like a String when assigned a String value, behaving like a Number when assigned a Number value, and so on.

AnyType data objects are included in ObjectPAL so you can use variables for basic data types without declaring them first. (Remember that it's better to declare variables whenever possible.)

Methods for the AnyType type

blank
dataType
fromHex
isAssigned
isBlank
isFixedType
toHex
unAssign
view

Print related ObjectPAL methods and examples

blank method/procedure

Returns a blank value.

Syntax

1. (Method) **blank ( )**
2. (Procedure) **blank ( )** AnyType

Description

blank generates a blank value to assign to a variable or field. A blank value is not the same as a numeric value of zero, but you can use Session type method **blankAsZero** to treat blank values as zeros in certain calculations. You can use the Session type method **isBlankZero** to find out whether Blank=Zero is on or off.

Example

{button ,AL(`OPAL_TYPE_ANYTYPE;OPAL_METH_ATISBL;',0,"Defaultoverview",)} **Related Topics**

blank example

The following example assumes that a form has a table frame bound to the *Lineitem* table and a button named *thisButton*. When a user presses *thisButton*, the code scans the Qty field in *Lineitem* and replaces non-blank values with blank values. This code is attached to the built-in **pushButton** method for *thisButton*:

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
  tc TCursor
endVar

if tc.attach(LINEITEM) then              ; attach tc to table frame
  tc.edit()                              ; edit the table frame
  scan tc for tc.Qty.isBlank() = False : ; look for non-blank Qty fields
    tc.Qty.blank()                       ; put a blank value in Qty
  endScan
  tc.endEdit()                           ; end edit mode
endif

endMethod
```

dataType method

Returns a string representing the data type of a variable.

Syntax

```
dataType ( ) String
```

Description

dataType returns a string representing the data type of a variable or expression: Binary, Currency, Date, DateTime, Graphic, Logical, LongInt, Memo, Number, OLE, Point, SmallInt, String, or Time. In comparison statements, you need to use one of the string values shown here. For example, the following is coded incorrectly because it compares String with string.

```
var s AnyType endVar
s = "This is a String data type."
msgInfo("Test", s.dataType() = "string")  ; displays False - should use "String"
```

 **Note**

- This method works for all ObjectPAL types, not just AnyType.

Example

{button ,AL(`OPAL_TYPE_ANYTYPE;OPAL_METH_ATISAS;',0,"Defaultoverview",)} **Related Topics**

dataType example

The following example assumes a form has a button and a graphic field named *bmpField*. The following code loads a DynArray with several different types of data and then uses **dataType** to display the data type of each value in the DynArray. This code is attached to the button's built-in **pushButton** method:

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
  mixedTypes DynArray[] AnyType
endVar


mixedTypes["Make"]  = "Ford"          ; String
mixedTypes["Model"] = "Cobra"         ; String
mixedTypes["Year"]  = 1969            ; SmallInt (not Date)
mixedTypes["Color"] = Black           ; LongInt - used here as a constant
mixedTypes["Photo"] = bmpField.value  ; Graphic


forEach element in mixedTypes    ; display a message for each element


  msgInfo("dataType(" + element + ")", dataType(mixedTypes[element]))


endForEach


endMethod
```

fromHex procedure

Converts a hexadecimal number to a decimal number.

Syntax

```
fromHex ( const value String ) LongInt
```

Description

fromHex converts a hexadecimal number to a decimal number. The value must range from 0x00000000 to 0xFFFFFFFF.

Example

{button ,AL(`OPAL_TYPE_ANYTYPE;OPAL_METH_ATTOHEX;',0,"Defaultoverview",)} **Related Topics**

fromHex example

In the following example, the **pushButton** method for a button named *convertHex* converts a hexadecimal string variable to a decimal number.

```
; convertHex::pushButton
method pushButton(var eventInfo Event)
    var
        s   String
        li  LongInt
    endVar

    ;Hexadecimal value to convert.
    s = "0x0756B5B3"
    s.view("Hex value to convert")
    li = fromHex(s)
    li.view("0x0756B5B3") ; Displays 123123123.
endMethod
```

isAssigned method

Reports whether a variable has been assigned a value.

Syntax

```
isAssigned ( ) Logical
```

Description

isAssigned returns True if the variable has been assigned a value; otherwise, it returns False.

Note

- This method works for all ObjectPAL types, not just AnyType.

Example

{button ,AL(`OPAL_TYPE_ANYTYPE;OPAL_METH_ATDTYP;OPAL_METH_ATUNASSIGN;',0,"Defaultoverview",)} **Related Topics**

isAssigned example

The following example uses **isAssigned** to test the value of *i* before assigning a value to it. If *i* has been assigned, this code increments *i* by one. The following code is attached in a button's Var window:

```
; thisButton::var
var
  i SmallInt
endVar
```

This code is attached to the button's built-in **pushButton** method:

```
; thisButton::pushButton
method pushButton(var eventInfo Event)


if i.isAssigned() then ; if i has a value
  i = i + 1            ; increment i
else
  i = 1                ; otherwise, initialize i to 1
endif
                       ; now show the value of i
message("The value of i is : " + String(i))


endMethod
```

isBlank method

Reports whether an expression has a blank value.

Syntax

isBlank ( ) Logical

Description

isBlank returns True if the expression has a blank value; otherwise, it returns False. Blank string values are denoted by "". Other blank values can be generated using **blank**. Note that blank values are not the same as 0, spaces (" "), or unassigned values.

Example

{button ,AL(`OPAL_TYPE_ANYTYPE;OPAL_METH_ATBLANK;',0,"Defaultoverview",)} Related Topics

isBlank example

The following example uses **isBlank** to test various values and displays the results in a dialog box. This code is attached to a button's **pushButton** method.

```
; thisButton::pushButton
method pushButton(var eventInfo Event)


msgInfo("Is the empty string blank?", isBlank(""))      ; True
msgInfo("Is a string of spaces blank?", isBlank("   ")) ; False
msgInfo("Is 5 a blank?", isBlank(5))                    ; False
msgInfo("Is blank blank?", isBlank(blank()))            ; True


endMethod
```

isFixedType method

Reports whether a variable's data type has been explicitly declared.

Syntax

```
isFixedType ( ) Logical
```

Description

isFixedType returns True if the variable has been declared using a **var...Endvar** block; otherwise, it returns False.

Example

{button ,AL(`OPAL_TYPE_ANYTYPE;OPAL_METH_ATDTYP;OPAL_METH_ATISAS;',0,"Defaultoverview",)}

Related Topics

isFixedType example

The following example demonstrates when **isFixedType** returns True. This code is attached to a button's built-in **pushButton** method.

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
  x SmallInt                  ; declare x
endVar


message(x.isFixedType())      ; displays True
sleep(2000)


testMe = 4                    ; testMe was not declared
message(testMe.isFixedType()) ; displays False


endMethod
```

toHex procedure

Converts a decimal number to a hexadecimal number.

Syntax

```
toHex ( const value LongInt ) String
```

Description

toHex converts a decimal number to a hexadecimal number.

Example

{button ,AL(`OPAL_TYPE_ANYTYPE;OPAL_METH_ATFROMHEX;',0,"Defaultoverview",)} **Related Topics**

toHex example

In the following example, the **pushButton** method for a button named *convertDecimal* converts a long integer value to a hexadecimal string.

```
; convertDecimal::pushButton
method pushButton(var eventInfo Event)
    var
        s   String
        li  LongInt
    endVar

    li = 123123123
    li.view("Value to convert")
    s = toHex(li)
    s.view("123123123") ; Displays 0x0756B5B3.
endMethod
```

unAssign method

Sets a variable's state to unAssigned.

Syntax

```
unAssign ( )
```

Description

unAssign sets a variable's state to unAssigned. The unAssigned state is not the same as a value of 0, nor is it the same as Blank.

Example

{button ,AL(`OPAL_TYPE_ANYTYPE;OPAL_METH_ATBLANK;OPAL_METH_ATISAS;',0,"Defaultoverview",)}

Related Topics

unAssign example

The following example demonstrates **unAssign**. This code is attached to a button's **pushButton** method.

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
  x AnyType
endVar

msgInfo("Is x assigned?", x.isAssigned()) ; displays False
x = 5
msgInfo("Is x assigned?", x.isAssigned()) ; displays True
x.unAssign()
msgInfo("Is x assigned?", x.isAssigned()) ; displays False

endMethod
```

view method

Displays the value of a variable in a dialog box.

Syntax

```
view ( [ const title String ] )
```

Description

view displays the value of a variable in a modal dialog box. ObjectPAL execution suspends until the user closes this dialog box. You have the option to specify, in *title*, a title for the dialog box. If you don't specify a title, the variable's data type appears.

The user can change the value displayed in a **view** dialog box as long as the data type is not an Array, DynArray, or Record. **view** cannot display Binary, Graphic, Memo, or OLE AnyTypes. The following table summarizes the AnyType variables that can be displayed, and those which the user can modify.

Type	Can be viewed	Can be modified
Binary	no	no
Currency	yes	yes
Date	yes	yes
DateTime	yes	yes
Graphic	no	no
Logical	yes	yes
LongInt	yes	yes
Memo	no	no
Number	yes	yes
OLE	no	no
Point	yes	yes
SmallInt	yes	yes
String	yes	yes
Time	yes	yes

Examples

{button ,AL(`OPAL_TYPE_ANYTYPE;OPAL_METH_ATISAS;OPAL_METH_ATUNASSIGN;',0,"Defaultoverview",)} **Related Topics**

view method examples

Example1	Beginner
Example2	Advanced

view example 1

The following example shows how you can determine whether the user clicked OK or Cancel (or closed the **view** dialog box another way, for example, by pressing ESC). When the user clicks OK, the value displayed in the dialog box is assigned to the variable. If the user closes the dialog box any other way, the value is not assigned.

The following code assigns an initial value to the variable and then tests to see if the variable has been changed. If the user has entered a valid value, the value is entered into the *ShipVia* field object in a form bound to the *Orders* table (assuming the form is in Edit mode).

This code is attached to a field object's built-in **arrive** method:

```
; shipViaFld::pushButton
method arrive(var eventInfo MoveEvent)
   var
      stShipVia,
      stPrompt  String
   endVar


   stPrompt  = "Enter Express or Regular."
   stShipVia = stPrompt


   stShipVia.view("Ship via:")


   if stShipVia = stPrompt then
         ; User closed the dialog box without changing the value.
         return
   else
         ; User entered a value and clicked OK.
         if stShipVia = "Express" or
               stShipVia = "Regular" then
               orders.shipVia.Value = stShipVia ; Or self.Value = stShipVia
         else
               msgStop("Stop", stPrompt)
         endIf
   endIf
endMethod
```

view example 2

The following example uses a **view** dialog box to prompt you for a date. If you enter a valid date, the code displays the day of the week for that date; otherwise, an error message is displayed.

```
; showDOW::pushButton
method pushButton(var eventInfo Event)
var
  theDate AnyType
  fullDays Array[7] String
endvar

fullDays[1] = "Sunday"
fullDays[2] = "Monday"
fullDays[3] = "Tuesday"
fullDays[4] = "Wednesday"
fullDays[5] = "Thursday"
fullDays[6] = "Friday"
fullDays[7] = "Saturday"

 ; initialize theDay variable
theDate = today()
 ; now show today's date in a dialog and prompt the user to enter a new date
theDate.view("Enter a Date")

 ; it's possible the user could enter an invalid date (like "Saturday")
 ; so this try..fail block attempts to convert theDate to a Date with
 ; dateVal() and if successful, displays the day of the week that
 ; theDate falls on
try
  msgInfo("Day of the week", String(theDate) + " falls on a\n" +
            fullDays[dowOrd(dateVal(theDate))])
onfail
  msgStop("Error!", theDate + " is not a valid date.")
endtry

endMethod
```

Application type

An Application variable provides a handle for working with the desktop window of the active Corel Paradox application. You can use an Application variable in your code to control the size, position, and appearance of the desktop, and change the working directory and the private directory at run time.

Although you can have more than one application running at the same time, Application objects can't communicate or operate on each other. An Application variable refers to the active Corel Paradox desktop only; you can, however, use Session variables to open multiple channels to the database engine (see the Session type).

Since there can be only one active application, to get an application handle, you must declare an Application type variable. While an Application variable is in scope, it serves as a handle to access the methods in the Application type. For instance, in the following example, an Application variable called *thisApp* is declared, and then used in the method's code.

```
; downSize::pushButton
method pushButton(var eventInfo Event)
var
  thisApp     Application
endVar
thisApp.maximize() ; Maximize the desktop.
endMethod
```

The following table displays the methods for the Application type, which are derived methods from the Form type.

Methods for the Application type

Form	Application
bringToTop	The Application type consists of derived methods from the Form type.
GetPosition	
getTitle	
hide	
isMaximized	
isMinimized	
isVisible	
maximize	
minimize	
setIcon	
setPosition	
setTitle	
show	
windowClientHandle	
windowHandle	

Array Type

An Array holds values (called *items* or *elements*) in *cells* similar to the way mail slots hold mail. An ObjectPAL array is one-dimensional, like a single row of slots, where each slot holds one item.

To use arrays in methods, you must declare them by specifying a name, size (number of items), and a data type for the items.

Notes

- In ObjectPAL, array items are counted beginning with 1, not with 0, as in some other languages.
- ObjectPAL also supports dynamic arrays. For more information, see the method and procedures for DynArray.

The following table displays the methods for the Array type, including several derived methods from the AnyType type.

Methods for the Array type

AnyType	Array
blank	**addLast**
dataType	**append**
isAssigned	**contains**
isBlank	**countOf**
isFixedType	**empty**
	exchange
	fill
	grow
	indexOf
	insert
	insertAfter
	insertBefore
	insertFirst
	isResizeable
	remove
	removeAllItems
	removeItem
	replaceItem
	setSize
	size
	view

Print related ObjectPAL methods and examples

addLast method

Inserts an item at the end of a resizeable array.

Syntax

```
addLast ( const value AnyType )
```

Description

addLast inserts *value* after the last item in a resizeable array. The array grows, if necessary, to make room for the new item. If you need to add more than one element to an array, use **grow** or **setSize** to allocate more space in the array rather than several **addLast** statements. For example, the following code uses **addLast** in a **for** loop to add 10 new elements to the *ar* array. Note that this use of **addLast** forces ObjectPAL to re-allocate space in the array 10 times; once each cycle through the loop.

```
for i from 11 to 20
  ar.addLast(i * 10)
endfor
```

The following code accomplishes the same as the previous code but executes faster because ObjectPAL allocates space only once:

```
ar.grow(10)    ; increase array size by 10 elements
for i from 11 to 20
  ar[i] = (i * 10)
endfor
```

Example

{button ,AL(`OPAL_TYPE_ARRAY;OPAL_METH_ARAPPE;OPAL_METH_ARINST;OPAL_METH_ARINAFT;OPAL_METH_ARINBEF;OPAL_METH_ARINFIRST;',0,"Defaultoverview",)} Related Topics

addLast example

The following example adds an element to a resizeable array each time *thisButton* is pressed. The **pushButton** method for *thisButton* increments the value of the newest element by 10 and displays the contents of the array in a **view** dialog box. The code immediately following should be attached in the Var window for *thisButton*:

```
; thisButton::Var
var
  ar Array[] SmallInt  ; declare ar as a resizeable array
  i  SmallInt          ; incrementing variable
endVar
```

The following code is attached to the built-in **pushButton** method for *thisButton*:

```
; thisButton::pushButton
method pushButton(var eventInfo Event)


                       ; initialize or increment i
i = iif(isAssigned(i), i + 10, 0)


if ar.size() = 0 then  ; true if this is the first time the button was pressed
  ar.setSize(0)        ; initialize size
endif


ar.addLast(i)          ; add another element to ar, and assign
                       ; the new element with the value of i


  ; display size of array in the title, and the value of
  ; each element in a view dialog box
ar.view("Size of ar array is " + strVal(ar.size()))


endMethod
```

append method

Appends the contents of one array to another.

Syntax

```
append ( const newArray Array[ ] String )
```

Description

append attaches the items of *newArray* to a resizeable array. The array grows to make room for the added items.

Example

{button ,AL(`OPAL_TYPE_ARRAY;OPAL_METH_ARALAS;OPAL_METH_ARINST;OPAL_METH_ARINAFT;OPAL_METH_ARINFIRST;',0,"Defaultoverview",)} Related Topics

append example

The following example creates two resizeable arrays, *addMe* and *baseArray*, and loads them with numeric values. The following example appends the *addMe* array to the *baseArray* array and then displays the results in a **view** dialog box. This code is attached to a button's built-in **pushButton** method:

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
   baseArray, addMe Array[] SmallInt
   i SmallInt
endVar


baseArray.setSize(3)
addMe.setSize(3)        ; now both arrays can store 3 values
for i from 1 to 3
   baseArray[i] = i     ; baseArray[1] = 1, [2] = 2, [3] = 3
   addMe[i] = (i + 3)   ;     addMe[1] = 4, [2] = 5, [3] = 6
endFor


baseArray.append(addMe) ; add the addMe array to baseArray
                        ; this grows baseArray to 6 elements


  ; now display the size of baseArray in the title of a view dialog
  ; and show baseArray elements within the dialog
baseArray.view("baseArray size: " + strVal(baseArray.size()))
endMethod
```

contains method

Searches the items of an array for a pattern of characters.

Syntax

```
contains ( const value AnyType ) Logical
```

Description

contains returns True if any item of an array exactly matches *value*; otherwise, it returns False.

Example

{button ,AL(`OPAL_TYPE_ARRAY;OPAL_METH_ARCNTO;',0,"Defaultoverview",)} **Related Topics**

contains example

The following example defines and loads a resizeable array named *dogs* when a form opens. Once the form's **open** method loads the array with dog names, the code displays the contents of the array in a dialog box. A button on the form contains code that uses the **contains** method to search the array for a particular name. If **contains** doesn't find the name, the built-in **pushButton** method attached to the button uses **insertFirst** to add the name to the top of the array.

The following code is attached to the form's Var window:

```
; thisForm::Var
var
  dogs Array[] String   ; resizeable array
endVar
```

The following code is attached to the form's built-in **open** method:

```
; thisForm::open
method open(var eventInfo Event)
if eventInfo.isPreFilter()
  then
    ;code here executes for each object in form
  else
    ;code here executes just for form itself


    dogs.setSize(4)     ; now dogs can store 4 values
    dogs[1] = "Bruno"   ; add some dog names
    dogs[2] = "Frodo"
    dogs[3] = "Yipper"
    dogs[4] = "Juneau"


      ; show the contents of the dogs array in a view dialog box
    dogs.view("dogs is initialized with these values")
endif
endMethod
```

This code is attached to the button's **pushButton** method:

```
; thisButton::pushButton
method pushButton(var eventInfo Event)


if dogs.contains("Bandit") = False then
  dogs.insertFirst("Bandit")  ; add new name to the top of the list
                              ; display contents of the array in a dialog box
  dogs.view("dogs size: " + strVal(dogs.size()))
else                          ; "Bandit" must already exist
  msgInfo("Once is enough", "The dogs array already contains Bandit.")
endif


endMethod
```

countOf method

Counts the occurrences of a value in an array.

Syntax

```
countOf ( const value AnyType ) LongInt
```

Description

countOf compares *value* to each item in an array and returns the number of exact matches or 0 if no match is found.

Example

{button ,AL(`OPAL_TYPE_ARRAY;OPAL_METH_ARCONT;',0,"Defaultoverview",)} Related Topics

countOf example

The following example contains code which should be attached to a button's **pushButton** method. It creates and loads a fixed-size array and then uses **countOf** to display the number of like values in the array:

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
  zoo Array[4] String
  i   SmallInt
endVar
for i from 1 to 3
  zoo[i] = "cat"       ; add three "cat" values
endFor
zoo[4] = "dog"         ; add one "dog" value


msgInfo("How many cats?", zoo.countOf("cat"))   ; displays 3
msgInfo("How many dogs?", zoo.countOf("dog"))   ; displays 1
msgInfo("How many apes?", zoo.countOf("ape"))   ; displays 0


endMethod
```

empty method

Removes all items from an array.

Syntax

```
empty ( )
```

Description

empty removes all items from an array. A fixed-size array stays the same size, and all items become unassigned. A resizeable array is reset to a size of 0.

Example

{button ,AL(`OPAL_TYPE_ARRAY;OPAL_METH_ARREMO;OPAL_METH_ARRALI;',0,"Defaultoverview",)}

Related Topics

empty example

The following example shows how **empty** functions for a fixed-size array. The code immediately following declares a fixed-size array in a form's Var window. This array is global to all objects on the form.

```
; thisForm::Var
Var
  ar Array[5] AnyType  ; declare a fixed-size array
endVar
```

The following code is attached to a button's **pushButton** method. When this button (*fillButton*) is pressed, the code assigns numeric values to each element in the *ar* array:

```
; fillButton::pushButton
method pushButton(var eventInfo Event)
ar[1] = 234    ; load the array with numbers
ar[2] = 356
ar[3] = 98
ar[4] = 989
ar[5] = 2341
               ; view the contents of the array
ar.view("Contents of the ar array")
endMethod
```

The following code is attached to a button's **pushButton** method. When this button (*emptyButton*) is pressed, the code empties the *ar* array and displays the contents of the array. Since *ar* is a fixed-size array, the number of elements does not change; there are still five elements, but each value becomes unassigned.

```
; emptyButton::pushButton
method pushButton(var eventInfo Event)
ar.empty()       ; empty the ar array
                 ; view the contents of the array
ar.view("Contents of the ar array")
endMethod
```

exchange method

Swaps the contents of two cells in an array.

Syntax

```
exchange ( const index1 LongInt, const index2 LongInt )
```

Description

exchange swaps the contents of the cells at *index1* and *index2* in an array.

Example

{button ,AL(`OPAL_TYPE_ARRAY;OPAL_METH_ARCONT;OPAL_METH_ARCNTO;OPAL_METH_ARINDX;',0,"Defaultoverview",)} **Related Topics**

exchange example

See the example for **indexOf**.

fill method

Fills an array with a value.

Syntax

```
fill ( const value AnyType )
```

Description

fill assigns a value to every item of an array.

Example

{button ,AL(`OPAL_TYPE_ARRAY;OPAL_METH_ARAPPE;OPAL_METH_ARINST;',0,"Defaultoverview",)}

Related Topics

fill example

The following example creates a fixed-size array and fills the array with String values. This code is attached to a button's **pushButton** method:

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
  myArray Array[4] String
endVar


myArray.fill("Hello")  ; fill myArray with Hello
myArray.view()         ; display four Hello's in a dialog


endMethod
```

grow method

Increases the size of a resizeable array.

Syntax

```
grow ( const increment LongInt )
```

Description

grow appends *increment* cells to a resizeable array or removes cells if the value of *increment* is negative. If you try to remove more cells than the array contains, an error occurs.

Example

{button ,AL(`OPAL_TYPE_ARRAY;OPAL_METH_ARALAS;OPAL_METH_ARINST;OPAL_METH_ARINFIRST;OPAL_METH_ARISDY;',0,"Defaultoverview",)} Related Topics

grow example

The following example uses **grow** to increase and decrease the size of a resizeable array. This code is attached to a button's **pushButton** method.

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
  ar Array[] SmallInt
endVar


ar.setSize(2)
ar[1] = 6
ar[2] = 123
message(ar.size()) ; displays 2
sleep(1000)
ar.grow(3)
message(ar.size()) ; displays 5
sleep(1000)
ar.grow(-3)
message(ar.size()) ; displays 2
sleep(1000)


endMethod
```

indexOf method

Returns the position of an item in an array.

Syntax

```
indexOf ( const value AnyType ) LongInt
```

Description

indexOf returns the index of the first occurrence of *value* in an array or 0 if an exact match is not found.

Example

{button ,AL(`OPAL_TYPE_ARRAY;OPAL_METH_ARCONT;OPAL_METH_ARCNTO;',0,"Defaultoverview",)}

Related Topics

indexOf example

The following example assumes a form has an undefined field object named *thisField*. When a user right-clicks on the field, a pop-up menu appears, offering a list of payment types. The item selected is inserted into the field. When the user next right-clicks the field, the last menu item selected is the first in the list of menu choices. The following code should be added in the Var window for *thisField*:

```
; thisField::Var
Var
  payArray  Array[5] String
  payMenu   PopUpMenu
endVar
```

The following code is attached to the **open** method for *thisField*. When the field first opens, values are assigned to the array that is used for the pop-up menu:

```
; thisField::open
method open(var eventInfo Event)
payArray[1] = "Check"      ; initialize array elements
payArray[2] = "Cash"
payArray[3] = "Visa"
payArray[4] = "MasterCard"
payArray[5] = "AmEx"
endMethod
```

The following code is attached to the **mouseRightUp** method for *thisField*. This code displays the pop-up menu and inserts the selection into *thisField*. The **indexOf** method is used here to get the ordinal value of the selected menu item; the selection is then moved, with the **exchange** method, to the beginning of the array.

```
; thisField::mouseRightUp
method mouseRightUp(var eventInfo MouseEvent)
var
  choiceIndex SmallInt
  choice      String
endVar


disableDefault                ; don't display the normal menu
payMenu.addArray(payArray)    ; add the array to the pop-up menu
choice = payMenu.show()       ; show the menu - assign selection to choice
self.value = choice           ; enter menu selection into field


  ; now prepare the pop-up menu for the next right click
payMenu.empty()                         ; empty the menu
choiceIndex = payArray.indexOf(choice) ; get the array index of the selection
payArray.exchange(choiceIndex, 1)      ; move the selection to the top
endMethod
```

insert method

Inserts one or more empty cells into an array.

Syntax

```
insert ( const index LongInt [ , const numberOfItems LongInt ] )
```

Description

insert add the number of empty cells specified by *numberOfItems* empty cells into a resizeable array. If *numberOfItems* is not specified, one cell is inserted. Indexes of subsequent items are increased by the number of inserted cells.

Example

{button ,AL(`OPAL_TYPE_ARRAY;OPAL_METH_ARINAFT;OPAL_METH_ARINBEF;',0,"Defaultoverview",)}

Related Topics

insert example

The following example inserts empty elements into a resizeable array at two locations and displays the results. This code is attached to a button's **pushbutton** method:

```
; thisbutton::pushbutton
method pushbutton(var eventinfo event)
var
  myArray Array[] SmallInt
endVar
myArray.setSize(20)   ; allocates space for 20 items
myArray.fill(1)       ; fills the array with 1's
myArray.insert(5)     ; inserts an empty cell at position 5
myArray.insert(12, 4) ; inserts 4 empty cells at position 12
myArray.view()
endMethod
```

insertAfter method

Inserts an item into an array after a specified item.

Syntax

```
insertAfter ( const keyItem AnyType, const insertedItem AnyType )
```

Description

insertAfter places *insertedItem* before the first occurrence of *keyItem* in a resizeable array. If *keyItem* is not found, *insertedItem* is not inserted, and the indexes do not change. If *insertedItem* is inserted, indexes of subsequent items increase by one.

Example

{button ,AL(`OPAL_TYPE_ARRAY;OPAL_METH_ARINST;OPAL_METH_ARINBEF;',0,"Defaultoverview",)}

Related Topics

insertAfter example

The following example loads a resizeable array, then uses **insertAfter** to insert a new element after an existing array element. This code is attached to a button's **pushButton** method:

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
  zoo Array[] String
endVar
zoo.setSize(0)
zoo.addLast("ape")      ; [1] = "ape"
zoo.addLast("cow")      ; [2] = "cow"
zoo.addLast("dog")      ; [3] = "dog"


zoo.insertAfter("ape", "bear")
  ; displays size: 4 in the title; zoo[ape, bear, cow, dog]
zoo.view("zoo size: " + strVal(zoo.size()))


endMethod
```

insertBefore method

Inserts an item into an array before a specified item.

Syntax

```
insertBefore ( const keyItem AnyType, const insertedItem AnyType )
```

Description

insertBefore searches a resizeable array for *keyItem* and inserts *insertedItem* at *keyItem*'s position. Indexes of *keyItem* (and subsequent items) are increased by one. If *keyItem* is not found, *insertedItem* is not inserted, and the indexes do not change.

Example

{button ,AL(`OPAL_TYPE_ARRAY;OPAL_METH_ARINST;OPAL_METH_ARINAFT;',0,"Defaultoverview",)}

Related Topics

insertBefore example

The following example adds an element to a resizeable array using **insertBefore**. This code is attached to a button's **pushButton** method:

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
  foodChain Array[] String
endVar


foodChain.grow(3)         ; start array out with 3 elements
foodChain[1] = "Hawk"
foodChain[2] = "Snake"
foodChain[3] = "Fly"


  ; insert an element - this increases the array to 4 elements
foodChain.insertBefore("Fly", "Frog")
  ; displays size: 4 in title; [Hawk, Snake, Frog, Fly]
foodChain.view("foodChain size: " + strVal(foodChain.size()))


endMethod
```

insertFirst method

Inserts an item at the beginning of an array.

Syntax

```
insertFirst ( const value AnyType )
```

Description

insertFirst inserts value at the beginning of a resizeable array. Indexes of subsequent items are increased by one.

Example

{button ,AL(`OPAL_TYPE_ARRAY;OPAL_METH_ARALAS;OPAL_METH_ARAPPE;OPAL_METH_ARINST;OPAL_METH_ARINAFT;OPAL_METH_ARINBEF;',0,"Defaultoverview",)} Related Topics

insertFirst example

The following example creates a resizeable array and then adds a new element to the beginning of the array. This code is attached to a button's built-in **pushButton** method:

```
method pushButton(var eventInfo Event)
var
  myZoo Array[] String
endVar
myZoo.setSize(2)   ; start the array with two elements
myZoo[1] = "lion"
myZoo[2] = "tiger"


                   ; insert an element at beginning of array -
                   ; this increases the array to three elements
myZoo.insertFirst("bear")
                   ; displays size: 3 in title; [bear, lion, tiger]
myZoo.view("myZoo size: " + strVal(myZoo.size()))


endMethod
```

isResizeable method

Reports whether an array can be resized.

Syntax

```
isResizeable ( ) Logical
```

Description

isResizeable returns True if an array can be resized; otherwise, it returns False.

Example

{button ,AL(`OPAL_TYPE_ARRAY;OPAL_METH_ARGROW;OPAL_METH_ARSIZ;',0,"Defaultoverview",)}

Related Topics

isResizeable example

The following example verifies whether a particular array can be resized before attempting to increase its size. This code is attached to a button's **pushButton** method:

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
  myArray Array[] String
endVar
if myArray.isResizeable() = True then  ; if array can be resized
  myArray.grow(5)                      ; add 5 cells to it
else
  msgStop("Problem", "Array cannot be resized.")
endif
endMethod
```

remove method

Removes one or more items from an array.

Syntax

```
remove ( const index SmallInt [ const numberOfItems SmallInt ] )
```

Description

remove deletes the number of items specified by *numberOfItems* items (or one item if *numberOfItems* is not specified) from an array. Indexes of subsequent items are decreased by *numberOfItems* (or one if *numberOfItems* is not specified).

Examples

{button ,AL(`OPAL_TYPE_ARRAY;OPAL_METH_ARINST;OPAL_METH_ARRITM;OPAL_METH_ARRALI;',0,"Defaultoverview",)} **Related Topics**

remove method examples

Example1	Removing one item from a resizeable array
Example2	Removing more than one item from a resizeable array

remove example 1

The following example removes a single item from a resizeable array. Note that it is common to use the **indexOf** method to determine which element you want to remove. This code is attached to a button's built-in **pushButton** method:

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
  myZoo Array[] String
endVar


myZoo.setSize(3)         ; start myZoo out with three elements
myZoo[1] = "lion"
myZoo[2] = "tiger"
myZoo[3] = "bear"


myZoo.remove(myZoo.indexOf("tiger"))  ; same as myZoo.remove(2)


                         ; title displays size: 2
                         ; dialog displays myZoo[lion, bear]
myZoo.view("myZoo size: " + strVal(myZoo.size()))


endMethod
```

remove example 2

The following example shows how to use **remove** to eliminate more than one element from a resizeable array. This code is attached to a button's **pushButton** method:

```
; thatButton::pushButton
method pushButton(var eventInfo Event)
var
  myNums Array[] SmallInt
  i      SmallInt
endVar


myNums.grow(9)      ; start myNums with nine elements
for i from 1 to 9   ; assign nine elements
   myNums[i] = i
endFor


                    ; displays myNums[1, 2, 3, 4, 5, 6, 7, 8, 9]
myNums.view("Before removing elements")
                    ; remove four items, starting with third element
myNums.remove(3, 4) ; myNums = [1, 2, 7, 8, 9]
                    ; displays myNums[1, 2, 7, 8, 9]
myNums.view("After removing elements")
endMethod
```

removeAllItems method

Removes all occurrences of an array item.

Syntax

```
removeAllItems ( const value AnyType )
```

Description

removeAllItems deletes all occurrences of *value* from an array. Indexes of subsequent items are decreased by one.

Example

{button ,AL(`OPAL_TYPE_ARRAY;OPAL_METH_ARREMO;OPAL_METH_ARRITM;',0,"Defaultoverview",)}

Related Topics

removeAllItems example

The following example shows how **removeAllItems** works with a resizeable array. The following code is attached to a button's built-in **pushButton** method:

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
   myZoo Array[] String
endVar
myZoo.setSize(5)
myZoo[1] = "ape"
myZoo[2] = "cow"
myZoo[3] = "pig"
myZoo[4] = "cow"
myZoo[5] = "lion"


  ; display current contents of array in a dialog
myZoo.view("Before removing elements")


  ; removes all occurrences of cow
myZoo.removeAllItems("cow")


  ; now,
  ; myZoo[1] = "ape"
  ; myZoo[2] = "pig"
  ; myZoo[3] = "lion"


  ; display new contents of array in a dialog
myZoo.view("After removing elements")


endMethod
```

removeItem method

Deletes a specified item from an array.

Syntax

```
removeItem ( const value AnyType )
```

Description

removeItem deletes the first occurrence of *value* from an array. Indexes of subsequent items are decreased by one.

Example

{button ,AL(`OPAL_TYPE_ARRAY;OPAL_METH_ARREMO;OPAL_METH_ARRALI;OPAL_METH_ARREIT;',0,"Defaultoverview",)} Related Topics

removeItem example

The following example uses **removeItem** to eliminate an item from a resizeable array. This code is attached to a button's built-in **pushButton** method:

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
  myZoo Array[] String
endVar


myZoo.setSize(4)
myZoo[1] = "ape"
myZoo[2] = "lion"
myZoo[3] = "tiger"
myZoo[4] = "lion"


  ; this displays [ape, lion, tiger, lion]
myZoo.view("Before removing a lion")


  ; remove first occurrence of "lion"
myZoo.removeItem("lion")


  ; this displays [ape, tiger, lion] in a dialog
myZoo.view("After removing a lion")


endMethod
```

replaceItem method

Overwrites an item in an array with another item.

Syntax

```
replaceItem ( const keyItem AnyType, const newItem AnyType )
```

Description

replaceItem searches an array for *keyItem*, and replaces the first occurrence of *keyItem* with *newItem*.

Example

{button ,AL(`OPAL_TYPE_ARRAY;OPAL_METH_ARRITM;',0,"Defaultoverview",)} **Related Topics**

replaceItem example

The following example replaces an item in a resizeable array and displays the original value and the results in a dialog box. This code is attached to a button's built-in **pushButton** method:

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
  foodChain Array[] String
endVar


foodChain.setSize(3)
foodChain[1] = "Shark"
foodChain[2] = "Elephant"
foodChain[3] = "Minnow"


  ; display contents of array in a dialog box
foodChain.view("Before replaceItem...")


foodChain.replaceItem("Elephant", "Tuna")
  ; display contents of array in a dialog box ([Shark, Tuna, Minnow])
foodChain.view("After replaceItem...")


endMethod
```

setSize method

Specifies the size of an array.

Syntax

```
setSize ( const size LongInt )
```

Description

setSize saves space for *size* items in a resizeable array. If **setSize** makes the array smaller, the array is truncated.

Example

{button ,AL(`OPAL_TYPE_ARRAY;OPAL_METH_ARGROW;OPAL_METH_ARISDY;',0,"Defaultoverview",)}

Related Topics

setSize example

The following example declares a resizeable array in the variable declaration section and then uses **setSize** to initialize the size of the array to three elements. The code fills each element of the array and then executes **setSize** again to resize the array to two elements. Making the array smaller (shown in a dialog box) eliminates of the third (and last) element. This code is attached to a button's built-in **pushButton** method:

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
  myArray Array[] SmallInt
endVar


myArray.setSize(3)          ; size is 3


myArray[1] = 123
myArray[2] = 2353
myArray[3] = 18


  ; display size: 3 in title; [123, 2353, 18] in a dialog box
myArray.view("myArray size: " + strVal(myArray.size()))


myArray.setSize(2)          ; size is 2- myArray[3] truncated


  ; display size: 2 in title; [123, 2353] in a dialog box
myArray.view("Now myArray size: " + strVal(myArray.size()))


endMethod
```

size method

Returns the number of items in an array.

Syntax

size ( ) LongInt

Description

size returns the number of items in an array, even if one or more elements are blank.

Example

{button ,AL(`OPAL_TYPE_ARRAY;OPAL_METH_ARGROW;OPAL_METH_ARSETSIZ;',0,"Defaultoverview",)}

Related Topics

size example

See the example for **setSize**.

view method

Displays the contents of an array in a dialog box.

Syntax

```
view ( [ const title String ] )
```

Description

view displays the contents of an array in a modal dialog box. ObjectPAL execution suspends until the user closes this dialog box. You have the option to specify, in *title*, a title for the dialog box. If you omit title, the title is Array.

Unlike many other data types, Array values displayed in a view dialog box can not be changed interactively. For more information, see AnyType.

Example

{button ,AL(`OPAL_TYPE_ARRAY;OPAL_METH_ARCONT;',0,"Defaultoverview",)} Related Topics

view example

The following example displays the contents of an array in a dialog box without a custom title and then with a custom title. This code is attached to a button's **pushButton** method:

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
  ar Array[]  SmallInt
  i           SmallInt
endVar


ar.setSize(10)
for i from 1 to 10
  ar[i] = i * 10
endfor

ar.view()        ; displays 10, 20, 30, etc (no title)
                 ; this displays "ar size: 10" in the title
ar.view("ar size: " + strVal(ar.size()))

endMethod
```

Binary type

A binary object (sometimes called a binary large object or BLOB) contains data that only a computer can read and interpret. An example of a binary object is a sound file; a human can't read or interpret the file in its raw form, but a computer can.

When you declare a Binary variable, you create a handle to a binary object. You can refer to this variable in your code to move binary data back and forth between a disk file and a binary field in a table or from a disk file or a table to a method or procedure.

The Binary type includes several derived methods from the AnyType type.

Methods for the Binary type

AnyType	Binary
blank	**clipboardErase**
dataType	**clipboardHasFormat**
isAssigne	**enumClipboardFormats**
isBlank	**readFromClipboard**
isFixedTy	**readFromFile**
	size
	writeToClipboard
	writeToFile

Print related ObjectPAL methods and examples

clipboardErase method

Clears the Windows Clipboard.

Syntax

```
clipboardErase ( )
```

Description

clipboardErase clears the Windows Clipboard on the user's system.

Example

{button ,AL(`OPAL_TYPE_BINARY;OPAL_METH_BIENUMCLIPBOARDFORMATS;OPAL_METH_BICLIPBOARDHASFORMAT;OPAL_METH_BIREADFROMCLIPBOARD;OPAL_METH_BIWRITETOCLIPBOARD;',0,"Defaultoverview",)} **Related Topics**

clipboardErase example

See the example for **clipboardHasFormat**.

clipboardHasFormat procedure

Reports whether a format name is on the Windows Clipboard.

Syntax

```
clipboardHasFormat ( const formatName String ) Logical
```

Description

clipboardHasFormat returns True if the format name *formatName* is on the Windows Clipboard on a user's system; otherwise, it returns False.

Example

{button ,AL(`OPAL_TYPE_BINARY;OPAL_METH_BIENUMCLIPBOARDFORMATS;OPAL_METH_BICLIPBOARDERASE;OPAL_METH_BIREADFROMCLIPBOARD;OPAL_METH_BIWRITETOCLIPBOARD;',0,"Defaultoverview",)} Related Topics

clipboardHasFormat example

In the following example, the **pushButton** method for a button named *clearClipboard* checks the Windows Clipboard for a Corel Form Object and if it is there, clears the Clipboard.

```
;btnClearClipboard::pushButton
method pushButton(var eventInfo Event)
   var
      b   Binary
   endVar


   if clipboardHasFormat("Corel Form Object") then
      b.clipBoardErase()
      message("Clipboard cleared")
   else
      message("Corel form object not on Clipboard")
   endIf
endMethod
```

enumClipboardFormats method

Creates an array listing the formats on the Windows Clipboard.

Syntax

```
enumClipboardFormats ( var formatNames Array[ ] String ) SmallInt
```

Description

enumClipboardFormats creates an array *formatNames* that lists the formats on the Windows Clipboard on the user's system. You must declare the array before you call this method.

Example

{button ,AL(`OPAL_TYPE_BINARY;OPAL_METH_BICLIPBOARDERASE;OPAL_METH_BICLIPBOARDHASFORMAT;OPAL_METH_BIREADFROMCLIPBOARD;OPAL_METH_BIWRITETOCLIPBOARD;',0,"Defaultoverview",)}

Related Topics

enumClipboardFormats example

The following example writes the Clipboard format names to an array named *ar*, then displays *ar* in a **view** dialog box.

```
;btnShowClipboard :: pushButton
method pushButton(var eventInfo Event)
   var
      b    Binary
      ar   Array[] String
   endVar


   b.enumClipboardFormats( ar )
   ar.view("Formats in Windows Clipboard")
endmethod
```

readFromClipboard method

Reads a binary object from the Clipboard.

Syntax

```
readFromClipboard ( const clipboardFormat String ) Logical
```

Description

readFromClipboard reads a binary object *clipboardFormat* from the Clipboard. If the Clipboard contains a Binary object that can be copied to a Binary variable, **readFromClipboard** returns True. If the Clipboard is empty or does not contain a valid Binary object, **readFromClipboard** returns False.

Example

{button ,AL(`OPAL_TYPE_BINARY;OPAL_METH_BIENUMCLIPBOARDFORMATS;OPAL_METH_BICLIPBOARDERASE;OPAL_METH_BICLIPBOARDHASFORMAT;OPAL_METH_BIWRITETOCLIPBOARD;',0,"Defaultoverview",)} Related Topics

readFromClipboard example

See the example for **writeToClipboard**.

readFromFile method

Reads data from a file and stores it in a Binary variable.

Syntax

```
readFromFile ( const fileName String ) Logical
```

Description

readFromFile reads binary data from the disk file named in *fileName*. This method returns True if successful; otherwise, it returns False.

Example

{button ,AL(`OPAL_TYPE_BINARY;OPAL_METH_BISIZE;OPAL_METH_BIWRITETOFILE;OPAL_TYPE_FILESYSTEM;',0,"Defaultoverview",)} Related Topics

readFromFile example

The following example declares a Binary variable *theSound*, reads binary data from a file into *theSound*, and then assigns the value of the variable to a Binary field in a table. Assume that SOUNDS.DB is a Corel Paradox table with the following structure: SoundName, A32; SoundData, and B.

```
; getFile::pushButton
method pushButton(var eventInfo Event)
var
  soundsTC TCursor
  theSound Binary
endVar
if theSound.readFromFile("noise.bin") then ; True if readFromFile succeeds
  if soundsTC.open("sounds.db") then
     soundsTC.edit()
     soundsTC.insertRecord()
     soundsTC.SoundName = "Noise"
     soundsTC.SoundData = theSound  ; put file contents in a binary field
     soundsTC.endEdit()
     soundsTC.close()
  endIf
endIf


endMethod
```

size method

Returns the number of bytes in a Binary variable.

Syntax

```
size ( ) LongInt
```

Description

size returns a value representing the number of bytes stored in a Binary variable.

Example

{button ,AL(`OPAL_TYPE_BINARY;OPAL_METH_BIREADFROMFILE;OPAL_METH_BIWRITETOFILE;OPAL_TYPE_FILESYSTEM;',0,"Defaultoverview",)} Related Topics

size example

The following example tests the **size** of each Binary field in a table. If there's enough free disk space, the code writes the data to a disk file. Assume that SOUNDS.DB is a Corel Paradox table with the following structure: SoundName, A32; SoundData, and B. This code is attached to a custom method named *writeBinFiles*:

```
method writeBinFiles()
var
  binVar    Binary
  fs        FileSystem
  soundsTC  TCursor
  freeSpace LongInt
endVar


if soundsTC.open("Sounds.db") then
  scan soundsTC for not isBlank(soundsTC.SoundData) :
    binVar = soundsTC.SoundData     ; binVar = SoundData field value
    freeSpace = fs.freeDiskSpace("B")
    if freeSpace > binVar.size() then     ; if there's room on B:
binVar.writeToFile(soundsTC.SoundName)     ; write binVar to file
    else          ; else the file won't fit on B:
      msgStop("Stop", "The disk in drive B: is full.")
      return
    endIf
  endScan
endif


endMethod
```

writeToClipboard method

Writes a binary object to the Clipboard.

Syntax

```
writeToClipboard ( const clipboardFormat String ) Logical
```

Description

writeToClipboard writes (copies) a binary object to the Windows Clipboard. Specify the Clipboard format to use with the parameter *clipboardFormat*. **writeToClipboard** returns True if successful; otherwise, it returns False.

Example

{button ,AL(`OPAL_TYPE_BINARY;OPAL_METH_BIENUMCLIPBOARDFORMATS;OPAL_METH_BICLIPBOARDERASE;OPAL_METH_BICLIPBOARDHASFORMAT;OPAL_METH_BIREADFROMCLIPBOARD;',0,"Defaultoverview",)} Related Topics

writeToClipboard example

In the following example, a form contains two buttons. The button named *btnStoreClip* stores the native portion of the Windows Clipboard in a file called *NATIVE.CLP*. The second button, *btnRetrieveClip,* retrieves the clip from the file and writes it to the Clipboard.

The following code is attached to *btnStoreClip*.

```
;btnStoreClip :: pushButton
method pushButton(var eventInfo Event)
   var
      b   Binary
   endVar


   if not b.readFromClipboard("Native") then
      msgInfo("Instructions", "First copy something to Clipboard.")
   endIf


   b.writeToFile("Native.clp")
endmethod
```

The following code is attached to *btnRetrieveClip*.

```
;btnRetrieveClip :: pushButton
method pushButton(var eventInfo Event)
   var
      b   Binary
   endVar


   if not b.readFromFile("Native.clp") then
      beep()
      message("File does not exist")
   endIf


   b.writeToClipBoard("Native")
endMethod
```

writeToFile method

Writes the data stored in a Binary variable to a disk file.

Syntax

```
writeToFile ( const fileName String ) Logical
```

Description

writeToFile copies the data stored in a Binary variable to the disk file specified in *fileName*. This method returns True if successful; otherwise, it returns False.

Example

{button ,AL(`OPAL_TYPE_BINARY;OPAL_METH_BIREADFROMFILE;OPAL_METH_BISIZE;OPAL_TYPE_FILESYSTEM;',0,"Defaultoverview",)} Related Topics

writeToFile example

The following example tests the size of each Binary field in a table. If there's enough free disk space, the code writes the data to a disk file. Assume that SOUNDS.DB is a Corel Paradox table with the following structure: SoundName, A32; SoundData, and B. This code is attached to a custom method named *writeBinFiles*:

```
method writeBinFiles()
var
  binVar    Binary
  fs        FileSystem
  soundsTC  TCursor
  freeSpace LongInt
endVar


if soundsTC.open("Sounds.db") then
  scan soundsTC for not isBlank(soundsTC.SoundData) :
    binVar = soundsTC.SoundData   ; binVar = SoundData field value
    freeSpace = fs.freeDiskSpace("B")
    if freeSpace > binVar.size() then         ; if there's room on B:
      binVar.writeToFile(soundsTC.SoundName) ; write binVar to file
    else                                      ; else the file won't fit
                                              ; on B:
      msgStop("Stop", "The disk in drive B: is full.")
      return
    endIf
  endScan
endif


endMethod
```

Currency type

Currency values can range from ± 3.4E-4930 to ± 1.1E4930 (precise to 18 decimal places). The number of decimal places displayed depends on the user's Control Panel settings. However, the values in a table are stored up to 18 decimal places.

The following table lists the methods for Currency type, including several derived methods from the Number and AnyType types.

Methods for the Currency type

AnyType	Number	**Currency**
blank	abs	**currency**
dataType	acos	
isAssigned	asin	
isBlank	atan	
isFixedType	atan2	
view	ceil	
	cos	
	cosh	
	exp	
	floor	
	fraction	
	fv	
	ln	
	log	
	max	
	min	
	mod	
	number	
	numVal	
	pmt	
	pow	
	pow10	
	pv	
	rand	
	round	
	sin	
	sinh	
	sqrt	
	tan	
	tanh	
	truncate	

Print related ObjectPAL methods and examples

currency procedure

Converts a value's data type to Currency.

Syntax

```
currency ( const value AnyType ) Currency
```

Description

currency casts *value* as a Currency.

Example

{button ,AL(`OPAL_TYPE_CURRENCY;OPAL_METH_NUNUMB;',0,"Defaultoverview",)} Related Topics

currency procedure examples

Example1	Performing a simple calculation
Example2	Performing more than one calculation

currency example 1

In the following example, a number is stored to a String variable and then cast as a Currency type for use in a calculation. The **pushButton** method for *showDouble* displays the type of the variable, and then calculates and displays the result of the string cast as Currency, and multiplied by two.

```
; showDouble::pushButton
method pushButton(var eventInfo Event)


var
  numstr   String
endVar


numStr = "12.34"
msgInfo("The data type of numStr is:", dataType(numStr))
; before multiplying numStr by two, it must be cast
; to a numeric type
msgInfo("Double " + numStr, currency(numStr) * 2)
endMethod
```

currency example 2

In the following example, the **pushButton** method for the *watchPrecision* button calculates a number using variables of the Number type, then performs the same calculation with the values cast as Currency. The result of the two calculations varies slightly.

```
; watchPrecision::pushButton
method pushButton(var eventInfo Event)


var
  x, y, z Number
endVar


x = 1.2 / 3.323        ; stores greatest precision
y = 4.9 / 7.3
z = 2.0 * x * y        ; calculates on full values
msgInfo("Result of Number calculation",
         format("W14.6", z))     ; displays .484790
x = Currency(1.2 / 3.323)        ; stores precision to 6th decimal place
y = Currency(4.9 / 7.3)
z = 2.0 * x * y                  ; calculates on 6 decimal precision values
msgInfo("Result of Currency calculation",
         format("W14.6", z))     ; displays .484791


endMethod
```

Database type

A Database variable provides a handle to a database (a directory). When you start a Corel Paradox application, Corel Paradox opens the default database (the working directory). The default database stores the path to the working directory. To work with tables stored in another database, declare a Database variable and use an **open** statement to create a handle to the database. You can also specify the full path to a table each time you wanted to use it, but code that uses Database variables is easier to maintain.

The following example demonstrates how to use **open** and an alias to specify which database to open:

```
var
   custInfo Database
endVar
; addAlias is defined for the Session type
addAlias("CustomerInfo", "Standard", "c:\\Corel\\Suite8\\Paradox\\tables\\custdata")
custInfo.open("CustomerInfo") ; opens the CustomerInfo database
                              ; CustomerInfo must be a valid alias
```

Corel Paradox now recognizes two databases: the default database and CustomerInfo. The *custInfo* variable is a *handle* to the CustomerInfo database and can be used in statements to refer to the CustomerInfo database. For example, suppose you have two files named ORDERS.DB (one in your working directory, and one in the CustomerInfo database), and you want to find out if these files are tables. The following example uses *custInfo* as a handle for the CustomerInfo database and tests ORDERS.DB:

```
var
   custInfo Database
endVar
addAlias("CustomerInfo", "Standard", "c:\\Corel\\Suite8\\Paradox\\tables\\custdata")
custInfo.open("CustomerInfo")


if isTable("orders.db") then        ; test ORDERS.DB in the default database
   msgInfo("Working directory", "ORDERS.DB is a table.")
endIf


if custInfo.isTable("orders.db") then   ; use custInfo as a handle for
                                        ; the CustomerInfo database
   msgInfo("CustomerInfo", "ORDERS.DB is a table.")
endIf
```

If you use **open** but don't specify a database, Corel Paradox assumes you want a handle for the default database. For example, the following syntax gives you a handle for the default database, which you can pass to a custom method that requires a database handle.

```
var defaultDb Database endVar
defaultDb.open() ; opens the default database
```

Methods for the Database type

beginTransaction
close
commitTransaction
delete
enumFamily
getMaxRows
isAssigned
isSQLServer
isTable
open
setMaxRows
rollBackTransaction
transactionActive

Print related ObjectPAL methods and examples

beginTransaction method

Starts a transaction.

Syntax

```
beginTransaction ( [ const isoLevel String ] ) Logical
```

Description

beginTransaction starts a transaction on a database that supports transactions, such as Corel Paradox, dBASE, and most SQL databases.

The optional argument *isoLevel* specifies an isolation level to use when transactions are supported on SQL databases. If you do not specify an isolation level, the highest (most isolated) isolation level supported by the server is used. The following table lists values for *isoLevel* from lowest to highest isolation level.

***isoLevel* value**	**Description**
DirtyRead	The transaction reads uncommitted changes made by other transactions.
ReadCommitted	Changes made by other transactions affect data read by the current transaction.
RepeatableRead	Data previously read in the current transaction is not affected by changes made by other transactions.

The **beginTransaction** method returns True if successful; otherwise, it returns False. While the transaction is active, statements that operate on tables associated with the specified database (except passthrough SQL statements) are included as part of the transaction. Only one transaction is allowed for each database.

Example

{button ,AL(`OPAL_TYPE_DATABASE;OPAL_METH_DBCOMMITTRANS;OPAL_METH_DBROLLBACKTRANS;OPAL_METH_DBTRANSACTIVE;',0,"Defaultoverview",)} Related Topics

beginTransaction example

The following example processes a withdrawal of cash from an automatic teller machine. The call to **beginTransaction** starts a transaction consisting of three operations: debiting the customer's account, debiting the cash on hand, and dispensing cash to the customer. The result of each operation is stored in a dynamic array (DynArray). When all of the operations are completed, this code checks each item in the DynArray and calls **commitTransaction** (if all items are True) or **rollbackTransaction** (if an item is False).

This example uses **beginTransaction**, **commitTransaction**, **rollbackTransaction**, **transactionActive**, **enumAliasNames**, and **getAliasProperty**.

```
method pushButton(var eventInfo Event)
   var
      db                   Database
      opResult             DynArray[] Logical
      Element              AnyType
      All_OK               Logical
      serverType,
      myAlias,
      custID               String
      aliasNamTC           TCursor
      xAmount              Currency
      xDate                Date
      xTime                Time
   endVar


   ; initialize variables
   myAlias = "ITCHY"
   custID = "RHALL001"
   xAmount = Currency(120.00)
   xDate = today() ; returns current date
   xTime = time()  ; returns current time


   ; use alias to get database handle to server
   if not db.open(myAlias) then
            errorShow("Could not open the database.")
      return ; exit the method
   endIf


   if db.transactionActive() then
      db.commitTransaction()       ; commit any previous transaction
   endIf


   db.beginTransaction()           ; begin a transaction


   ; execute the operations for this transaction
   ; debitAccount, debitCashOnHand, and dispenseCash
   ; are custom procs assumed to be defined elsewhere
   ; after calling debitAccount and debitCashOnHand, the code
   ; calls transactionActive to check the transaction status
   ; before calling dispenseCash


   opResult["Debit customer account"] =
            debitAccount(custID, xAmount)
   opResult["Debit cash on hand"] =
            debitCashOnHand(xAmount, xDate, xTime)


   ; the following if...then...else block is not required
   ; it's included to show one way to use transactionActive


   if db.transactionActive() then  ; make sure everything is OK
      msgInfo("Transaction Status", "In a Transaction")
   else
      errorShow("NOT in a Transaction")
      return
   endIf
```

```
   opResult["Dispense cash"] = dispenseCash(xAmount)

   All_OK = True                ; initialize to True

   forEach element in opResult   ; Check operation results
      if opResult[element] = False then
         All_OK = False
         quitLoop
      endIf
   endForEach

   ; inform user of transaction status
   if All_OK then
      if db.commitTransaction() then
         msgInfo("Transaction Status","Transaction committed.")
      else
         errorShow("Transaction NOT committed")
      endIf
   else
      if msgQuestion("Transaction failed",
                     "View results?") = "Yes" then
         opResult.view("Operation results")
      endIf

      if db.rollbackTransaction() then
         msgInfo("Transaction Status",
                 "Transaction rolled back.")
      else
         errorShow("Transaction NOT rolled back.")
      endIf
   endIf

endMethod
```

close method

Closes a database.

Syntax

close ( ) Logical

Description

close disassociates a Database variable and a database, making the variable unassigned. **close** returns True if it succeeds; otherwise, it returns False.

Example

{button ,AL(`OPAL_TYPE_DATABASE;OPAL_METH_DBOPEN;',0,"Defaultoverview",)} **Related Topics**

close example

The following example opens the database with the alias *someTables*. If the *Orders* table doesn't exist in *someTables*, this code closes *someTables* and opens another database with the alias *moreTables*. This code assumes that both aliases have been defined elsewhere and are valid.

```
; sumButton::pushButton
method pushButton(var eventInfo Event)
var
  db Database
  tc TCursor
endVar
db.open("someTables")              ; open the database alias someTables
if db.isTable("Orders.db") then    ; if Orders.db is in the database,
  tc.open("Orders.db", db)         ; open a TCursor for it
                                   ; calculate the total balance due
  msgInfo("Balance Due", tc.cSum("Balance Due"))
else
  db.close()                       ; close someTables database
  db.open("moreTables")            ; and open another one
  if db.isTable("Orders.db") then
    tc.open("Orders.db", db)
    msgInfo("Balance Due", tc.cSum("Balance Due"))
  endIf
endIf
endMethod
```

commitTransaction method

Commits all changes within a transaction.

Syntax

commitTransaction ( ) Logical

Description

commitTransaction commits all changes made within a transaction on a database that supports transactions, such as Corel Paradox, dBASE, and most SQL databases.

commitTransaction returns True if successful; otherwise, it returns False. This method does not check the results of the operations in the transaction; instead, you must evaluate the results and decide whether to commit the transaction or roll it back.

Example

{button ,AL(`OPAL_TYPE_DATABASE;OPAL_METH_DBBEGINTRANS;OPAL_METH_DBROLLBACKTRANS;OPAL_METH_DBTRANSACTIVE;',0,"Defaultoverview,")} Related Topics

commitTransaction example

See the **beginTransaction** example.

delete method/procedure

Deletes a table from a database.

Syntax

```
1. delete ( const tableName String [ , const tableType String ] ) Logical
2. delete ( const tableVar Table ) Logical
```

Description

delete removes a table and any associated index files or table view files from the database without asking for confirmation. If you use Syntax 1 and the file extension is not standard or not supplied, you can use the optional argument *tableType* to specify the type of table to delete (e.g., Corel Paradox or dBASE). If *tableType* is not specified or not standard, **delete** removes the Corel Paradox table. If you use Syntax 2, you can use the argument *tableVar* to specify a Table variable. This method uses the name and type of table described by the Table variable, but does not use its database association. In either case, the deletion can't be undone.

This method returns True if the table is successfully deleted; otherwise, it returns False. If the table is open, **delete** fails.

Example

{button ,AL(`OPAL_TYPE_DATABASE;OPAL_METH_DBISTA;',0,"Defaultoverview",)} Related Topics

delete example

In the following example, the **pushButton** method for *delTable* deletes a table from the database with the alias *megaData*.

```
; delTable::pushButton
method pushButton(var eventInfo Event)
var
  myDb Database
  tableName String
endVar
tableName = "OldTable.dbf"
myDb.open("megadata")
if isTable(tableName) then
  myDb.delete(tableName, "dBASE") ; removes OldTable.dbf from megadata
endif
endMethod
```

enumFamily method/procedure

Lists the files in a table family.

Type

Database

Syntax

```
enumFamily ( var members DynArray[ ] String, const tableName String ) Logical
```

Description

enumFamily lists the files in the table family of the table *tableName*. It assigns values to a dynamic array, or DynArray, named *members* that you pass as an argument. The value of *tableName* must include a file extension if the table name includes one. For example, if you specify ORDERS as the value, this method *does not* list the table family for ORDERS.DB; instead, **enumFamily** looks for a table named ORDERS.

The DynArray's indexes represent the full filenames of the table family members, and the corresponding value is one of the following strings:

Blobfile

Form

Index

Report

SecondaryIndex

SecondaryIndex2

Table

Unknown

ValCheck

Example

{button ,AL(`OPAL_TYPE_DATABASE;OPAL_METH_DBISTA;OPAL_METH_TCFAM;',0,"Defaultoverview",)}

Related Topics

enumFamily example

The following example copies the family information from the *Orders* table to a dynamic array *dyn.* A **forEach** loop then displays each element of the family information in a dialog box.

```
;btnFamilyInfo :: pushButton
method pushButton(var eventInfo Event)
   var
      dyn      DynArray[] String
      sElement   String
   endVar


   enumFamily(dyn, "ORDERS.DB")

   forEach sElement in dyn
      msgInfo(sElement, dyn[sElement])
   endForEach
   ; You could also do dyn.view().
endmethod
```

getMaxRows method

Retrieves the setting of **setMaxRows** (the maximum number of rows that are returned from an SQL server).

Syntax

```
getMaxRows ( const maxRows LongInt ) Logical
```

Description

getMaxRows retrieves the setting on the maximum number of rows that are returned from an SQL server in response to a query. Use **setMaxRows** to set the maximum number of returns.

Example

{button ,AL(`OPAL_TYPE_DATABASE;OPAL_METH_DBISSQLSERVER;',0,"Defaultoverview",)} Related Topics

getMaxRows example

The following example puts a 1000 record limit on the query if the maximum is set to less than 1000.

```
var myQBE Query
endvar
   if getMaxRows() < 1000 then
      setMaxRows(1000)
   endif
   myQBE = Query
         Customer    |Customer No |Name  |
                     | Check      |A..  |
   endQuery
```

isAssigned method

Reports whether a Database variable has been assigned a value.

Syntax

isAssigned ( ) Logical

Description

isAssigned returns True if the Database variable has been assigned a value; otherwise, it returns False.

Example

{button ,AL(`OPAL_TYPE_DATABASE;OPAL_METH_DBISTA;',0,"Defaultoverview",)} **Related Topics**

isAssigned example

In the following example, a form has been created with an unassigned field named coRating and a button named *showRating*. The code attached to *showRating*'s **pushButton** method uses **isAssigned** to determine whether the Database variable *db* is assigned. If a value has now been assigned to the variable *db*, a database alias is assigned to the Database variable. Once the variable is defined, the code opens a TCursor for the *NewCust* table contained in the database. The TCursor locates a value in the Company field, then displays that company's credit rating in the *coRating* field on the form. The following code is attached to the **pushButton** method for *showRating*:

```
; showRating::pushButton
method pushButton(var eventInfo Event)
var
  db Database
  tc TCursor
endVar


if not isAssigned(db) then
  addAlias("myTables", "Standard", "c:\\Corel\\Suite8\\Paradox\\myTables")
  db.open("myTables")
endif


tc.open("NewCust.dbf", db)
if tc.locatePattern("Company", "Thompson's..") then
  coRating.value = tc.Rating
else
  message("Error", "Thompson's.. not found.")
endif


endMethod
```

isSQLServer method

Reports whether a Database is opens on an SQL server.

Syntax

isSQLServer ( ) Logical

Description

isSQLServer returns True if the Database variable is open on an SQL server; otherwise, it returns False.

Example

{button ,AL(`OPAL_TYPE_DATABASE;OPAL_METH_DBISASSIGN;OPAL_METH_DBISTA;',0,"Defaultoverview",)} Related Topics

isSQLServer example

In the following example, a Database variable is opened on an alias. The code then determines if the Database variable points to an SQL server, and displays the results.

```
; showRating::pushButton
method pushButton(var eventInfo Event)
   var
      db Database
   endVar


   db.open(":fred:")


   if db.isSQLServer() then
      msgInfo(":FRED:", "Is on a SQL server.")
   else
      msgInfo(":FRED:", "Is not on a SQL server.")
   endIf


endMethod
```

isTable method/procedure

Reports whether a database contains a table.

Syntax

```
1. isTable ( const tableName String [ , const tableType String ] ) Logical
2. isTable ( const tableVar Table ) Logical
```

Description

isTable returns True if the specified table is found in the database; otherwise, it returns False.

If you use Syntax 1, you can specify the table's name and type in the arguments *tableName* and *tableType*. If you use Syntax 2, you can specify a Table variable in *tableVar.* This method uses the table name and type described by the Table variable, and not the database association.

 **Example**

{button ,AL(`OPAL_TYPE_DATABASE;OPAL_METH_DBISASSIGN;',0,"Defaultoverview",)} **Related Topics**

isTable example

The following example uses **isTable** to determine whether the *Orders* table exists in a given database. The code is attached to the built-in **pushButton** method for *thisButton*.

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
  db        Database
  testMe    String
  testMeToo Table
  myTable   TableView
endVar


db.open()                         ; opens the default database
testMe = "Orders.db"
if db.isTable(testMe) then
  myTable.open(testMe)
else
  message(testMe, " is not a table!")
endIf


testMeToo.attach("sales.db")
if testMeToo.isTable() then
  tot = testMeToo.cSum("Total sales")
  msgInfo("total sales:", tot)
endif
endMethod
```

open method/procedure

Opens a database.

Syntax

```
1. open ( ) Logical

2. open ( const aliasName String ) Logical

3. open ( const ses Session ) Logical

4. open ( const aliasName String, const ses Session ) Logical

5. open ( [ const aliasName String, ] [ const ses Session, ]
             [ const parms DynArray ] ) Logical
```

Description

open opens a database. In Syntax 1, where no arguments are given, **open** opens the default database. In Syntax 2, you specify in *aliasName* a database to open in the current session. Syntax 3 opens the default database in the session specified in *ses*. Use Syntax 4 to open a specified database in a specified session. In Syntax 5, the *parms* argument represents a list of parameters and values to use when opening a database on an SQL server. The items in the parameter list correspond to the fields in the Alias Manager dialog box for a given alias. The items vary depending on the server you're connecting to; see your server documentation for more information.

open returns True if it opens the specified database; otherwise, it returns False.

Notes

- If you use Syntax 2, 4, or 5, *aliasName* must be a valid alias in the current session or the *ses* session. The colons around the alias name are optional.
- Syntaxes 3, 4, and 5 require that a valid session variable has been opened; the current session is assumed in Syntaxes 1 and 2.
- When you use Syntax 5, the settings in the *parms* dynamic array override previously set values, both in code and interactively. For example, if the OPEN MODE parameter was previously set to READ/WRITE, the following statement would set it to READ ONLY when you open the database.

  dbParmsDA["OPEN MODE"] = "READ ONLY"

- When you use *parms* to specify parameters, the Alias Manager dialog box does not open.

Example

{button ,AL(`OPAL_TYPE_DATABASE;OPAL_METH_DBCLOS;OPAL_TYPE_SESSION;',0,"Defaultoverview",)}

Related Topics

open example

For the following example, the **pushButton** method for *thisButton* opens four databases in the current session.

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
  dDb, myDb, pDb, rDb  Database
  dbParmsDA            DynArray[] AnyType
  currSes              Session
endVar


currSes.open()          ; get a handle to the current session


dDb.open()              ; associate dDb with the default database
myDb.open("custInfo")   ; associate myDb with the Custinfo database
                        ; (custInfo is an alias defined elsewhere)
pDb.open("PRIV")        ; associate pDb with the Private directory


; specify parameters for SQL database
dbParmsDA["OPEN MODE"] = "READ/WRITE"
dbParmsDA["Password"]  = "tycobb"


rDb.open("remote", currSes, dbParmsDA) ; (remote is an alias defined elsewhere)
endMethod
```

rollBackTransaction method

Rolls back or undoes all changes within a transaction, on a server that supports transactions.

Syntax

```
rollbackTransaction ( ) Logical
```

Description

rollbackTransaction undoes the effects of all operations within a transaction. This method returns True if successful; otherwise, it returns False.

Example

{button ,AL(`OPAL_TYPE_DATABASE;OPAL_METH_DBBEGINTRANS;OPAL_METH_DBCOMMITTRANS;OPAL_METH_DBTRANSACTIVE;',0,"Defaultoverview",)} Related Topics

rollBackTransaction example

See the **beginTransaction** example.

setMaxRows method

Sets the maximum number of rows that can be retrieved by one query.

Syntax

```
setMaxRows ( const maxRows LongInt ) Logical
```

Description

setMaxRows sets the maximum number of rows that can be returned from an SQL server in response to a single query. The argument *maxRows* is a long integer that specifies the maximum number of rows returned. **setMaxRows** returns True if the maximum number of rows specified by *maxRows* is successfully set; otherwise, it returns False.

SetMaxRows resemble the Borland Database Engine (BDE) configuration option MAX ROWS. MAX ROWS is set in the BDE Configuration file's DB OPEN section and sets the maximum number of rows that the SQL driver will attempt to fetch for any single SQL statement that is sent to the server. If a request exceeds the maximum specified by MAX ROWS, Corel Paradox generates a DBIERR_ROWFETCHLIMIT error.

Notes

- The maximum specified with the **setMaxRows** method can exceed that specified by the MAX ROWS BDE configuration option.
- If no **setMaxRows** method is issued or if the *maxRows* argument is set to -1, Corel Paradox imposes no limit on rows. If present, the BDE MAX ROWS limit is imposed.

Example

{button ,AL(`OPAL_TYPE_DATABASE;OPAL_METH_DBISSQLSERVER;',0,"Defaultoverview",)} Related Topics

setMaxRows example

The following example puts a 1000 record limit on the query if the maximum is set to less than 1000. This example assumes the database is open.

```
var
    myQBE Query
    myDatabase Database
endvar
myDatabase.open("Work")
   if database.getmaxrows()<1000 then
then
      database.setmaxrows(1000)
endif
myQBE = Query
         Customer      |Customer No |Name  |
                       | Check      |A..  |
   endQuery
```

transactionActive method

Reports whether a transaction is active in the specified database.

Syntax

transactionActive ( ) Logical

Description

transactionActive reports whether a transaction is active in the specified database. Because Corel Paradox allows only one active transaction for each database, call **transactionActive** to determine whether a transaction is active before beginning a transaction.

Example

{button ,AL(`OPAL_TYPE_DATABASE;OPAL_METH_DBBEGINTRANS;OPAL_METH_DBCOMMITTRANS;OPAL_METH_DBROLLBACKTRANS;',0,"Defaultoverview",)} Related Topics

transactionActive example

See the **beginTransaction** example.

Date type

In ObjectPAL, you can represent Date values in either month/day/year, day-month-year, or day.month.year format. Dates must be explicitly declared. For example, the following code assigns the date December 21 1997 to *d*.

```
var
  d Date
endVar
d = date("12/21/1997")
```

If you omit the quotes around the Date value ObjectPAL divides the values.

The Date type includes methods defined for the AnyType type and the DateTime type. For more information, see AnyType and DateTime.

Date values are formatted by the **formatSetDateDefault** method (System type), or by ObjectPAL formatting statements.

Dates from the 20th century can be specified using two digits for the year, as in

```
myDay = date("11/09/59")        ; November 9, 1959
```

Dates from the 2nd to the 10th centuries must include three digits of the year (e.g., 12/17/243). Dates from the 11th to the 19th centuries must have four digits (e.g.,12/17/1043). The year cannot be omitted completely. Corel Paradox treats all dates in the B.C. era as leap years.

The Date type includes several derived methods from the DateTime and AnyType types. The Date type also includes several methods defined for the DateTime type. For more information, see DateTime.

Methods for the Date type

AnyType	DateTime	Date
blank	day	**date**
dataType	daysInMonth	**dateVal**
isAssigned	dow	**today**
isBlank	dowOrd	
isFixedType	doy	
view	isLeapYear	
	month	
	moy	
	year	

Print related ObjectPAL methods and examples

date method

Returns a Date value.

Syntax

```
1. date ( const value AnyType ) Date
2. date ( ) Date
3. date ( month SmallInt, day SmallInt, year SmallInt ) Date
```

Description

date casts **value** as a date. If the date specified by **value** is invalid, the method fails. If you do not define **value**, **date** returns the current system date as a Date value.

If you use Syntax 3, the month ranges from 1 to 12. The day range depends on the month and can range from 1 to 28 or 31. The year can range from -9999 to 9999 all four digits must be used (i.e., 1997). An error is returned if a value does not lie within the required range. For example, specifying 40 for the day value generates a Bad Day Specification error message.

Example

{button ,AL(`OPAL_TYPE_DATE;OPAL_METH_DADAVA;',0,"Defaultoverview",)} **Related Topics**

date example

The following example casts a String value as a date, uses the Date value in a calculation, and displays the result in a dialog box:

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
  s String
  d Date
endVar


s = "11/11/99"   ; s is a String value
d = date(s) + 7  ; convert String type to a Date type and add 7 days
                 ; and add 7 days


d.view()         ; show value of d in a dialog box (11/18/99)
                 ; dialog box title displays "Date"
endMethod
```

dateVal procedure

Returns a specified value as a date.

Syntax

```
dateVal ( const value AnyType ) Date
```

Description

dateVal returns a specified value as a date.

Example

{button ,AL(`OPAL_TYPE_DATE;OPAL_METH_DADATE;',0,"Defaultoverview",)} Related Topics

dateVal example

In the following example, the **pushButton** method for a button uses **dateVal** to convert a String value into a Date value. The result is displayed in a dialog box.

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
  s String
  d Date
endVar


s = "11/11/99"   ; s is a String value
d = dateVal(s)   ; d holds the date equivalent of s


d.view()         ; show value of d in a dialog box (11/11/99)
                 ; dialog box title displays "Date"
endMethod
```

today procedure

Returns the current date.

Syntax

today ( ) Date

Description

today returns the current date, as displayed on your system clock/calendar.

Example

{button ,AL(`OPAL_TYPE_DATE;OPAL_METH_DADATE;OPAL_METH_DTDAY;OPAL_METH_DTMONTH;OPAL_METH_DTYEAR;',0,"Defaultoverview",)} Related Topics

today example

The following example displays the current date in a dialog box:

```
; CurrentDate::pushButton
method pushButton(var eventInfo Event)
msgInfo("Today's Date", today())       ; displays the current date
endMethod
```

DateTime type

A DateTime variable stores data in the form hour-minute-second-millisecond year-month-day. DateTime values are used only in ObjectPAL calculations; you cannot store a DateTime value in a Corel Paradox table. DateTime values must be explicitly declared. For example, in the following statements, the time assigned to the DateTime variable *dt* is 10 minutes and 40 seconds past eleven o'clock and the date is December 21, 1997. DateTime values must be enclosed in quotation marks.

```
var dt DateTime endVar
dt = DateTime("11:10:40 am 12/21/97")
```

You can use the following characters as separators in your DateTime specifications: blank, tab, space, comma, hyphen, slash, period, colon, and semicolon. DateTime values are formatted by the **formatSetDateTimeDefault** procedure (System type) or by ObjectPAL formatting statements.

You must specify a DateTime value completely, including all of the fields. Specify a value of zero for empty fields.

For more information, see the methods and procedures for the Date type and the Time type.

The following table lists the methods of the DateTime type, including several derived methods from the AnyType type.

Methods for the DateTime type

AnyType	DateTime
blank	**dateTime**
dataType	**day**
isAssigned	**daysInMonth**
isBlank	**dow**
isFixedType	**dowOrd**
view	**doy**
	hour
	isLeapYear
	milliSec
	minute
	month
	moy
	second
	year

Print related ObjectPAL methods and examples

dateTime method

Returns a DateTime value.

Syntax

```
1. dateTime ( const value AnyType ) DateTime
2. dateTime ( ) DateTime
```

Description

dateTime casts *value* as a DateTime data type. If *value* is not supplied, **dateTime** returns the system date and time as a DateTime value.

Example

{button ,AL(`OPAL_TYPE_DATETIME;OPAL_METH_DTDAY;OPAL_METH_DTHOUR;OPAL_METH_DTMONTH;OPAL_METH_DTYEAR;',0,"Defaultoverview",)} Related Topics

dateTime example

The following statements assign date and time values to the DateTime variable *dt.* The time is 10 minutes and 40 seconds past eleven o'clock and the date is December 21, 1997. This code assumes the current date and time format is in the form hh:mm:ss am/pm mm/dd/yy. DateTime values must be enclosed in quotation marks.

```
var dt DateTime endVar
dt = dateTime("11:10:40 am 12/21/97")
```

You can use the following characters as separators in your DateTime specifications: blank, tab, space, comma, hyphen, slash, period, colon, and semicolon. DateTime values are formatted by **formatSetDateTimeDefault** (System type) or by ObjectPAL formatting statements.

You must specify a DateTime value completely, including all of the fields. Specify a value of zero for empty fields.

day method

Extracts the day of the month from a date.

Syntax

```
day ( ) SmallInt
```

Description

day extracts the day of the month from a DateTime value and returns a value between 1 and 31. If the DateTime value is invalid, the method fails.

Example

{button ,AL(`OPAL_TYPE_DATETIME;OPAL_METH_DTDOW;OPAL_METH_DTDOWORD;OPAL_METH_DTDOY;OPAL_METH_DTMONTH;OPAL_METH_DTMOY;OPAL_METH_DTYEAR;',0,"Defaultoverview",)} Related Topics

day example

The following example uses a button's **pushButton** method to display the current day of the month in a dialog box. This code assumes the current date and time format is in the form hh:mm:ss am/pm mm/dd/yy.

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
  theDay DateTime
endVar
theDay = DateTime("12:00:00 am 12/22/92")


  ; displays 22 in a dialog box
msgInfo("Day of the month", theDay.day())


endMethod
```

daysInMonth method

Returns the number of days in a month.

Syntax

daysInMonth ( ) SmallInt

Description

daysInMonth returns the number of days in the month specified in a DateTime value. If the DateTime value is invalid, the method fails.

Example

{button ,AL(`OPAL_TYPE_DATETIME;OPAL_METH_DTDAY;OPAL_METH_DTDOW;OPAL_METH_DTDOWORD;OPAL_METH_DTMOY;',0,"Defaultoverview",)} Related Topics

daysInMonth example

The following example uses a button's **pushButton** method to display the number of days in February 1992 in a dialog box. This code assumes the current date and time format is in the form hh:mm:ss am/pm mm/dd/yy.

```
; FebDays::pushButton
method pushButton(var eventInfo Event)
var
  daysInFeb SmallInt
endVar
daysInFeb = daysInMonth(DateTime("5:15:35 AM 2/1/92"))
msgInfo("Number of days", "There are " + String(daysInFeb) +
            " days in February 1992")


  ; displays "There are 29 days in February 1992" in a dialog box
  ; (1992 is a leap year)
endMethod
```

dow method

Returns the day of the week.

Syntax

```
dow ( ) String
```

Description

dow returns the first three letters of the day in a specified DateTime value. If the DateTime value is not valid, the method fails.

Example

{button ,AL(`OPAL_TYPE_DATETIME;OPAL_METH_DTDATETIME;OPAL_METH_DTDAY;OPAL_METH_DTDOWORD;OPAL_METH_DTDOY;OPAL_METH_DTMOY;',0,"Defaultoverview",)} **Related Topics**

dow example

The following example displays the day of week from a specified DateTime value. This code assumes the current date and time format is in the form hh:mm:ss am/pm mm/dd/yy.

```
; showDay::pushButton
method pushButton(var eventInfo Event)
var
  theDate DateTime
endVar


theDate = DateTime("11:20:15 pm 3/9/93")


  ; displays "Tue" in a dialog box
msgInfo("Day of Week", strVal(theDate) + " falls on a " + dow(theDate))


endMethod
```

dowOrd method

Returns the number representing a specified day's position in the week.

Syntax

```
dowOrd ( ) SmallInt
```

Description

dowOrd returns an integer from one to seven representing a specified day's position in the week. Sunday is day one, Monday is day two, and so on. If the DateTime value given is invalid, the method fails.

Example

{button ,AL(`OPAL_TYPE_DATETIME;OPAL_METH_DTDATETIME;OPAL_METH_DTDAY;OPAL_METH_DTDOW; OPAL_METH_DTDOY;OPAL_METH_DTMOY;',0,"Defaultoverview",)} Related Topics

dowOrd example

The following example displays the day of the week by name rather than by abbreviation or number. This code uses **dowOrd** to retrieve the appropriate subscript of a fixed array, then displays the value of the array element in a dialog box. This code is attached to the **pushButton** method for the *fullDay* button. This example assumes the current date and time format is in the form hh:mm:ss am/pm mm/dd/yy.

```
; fullDay::pushButton
method pushButton(var eventInfo Event)
var
  fullDays Array[7] String
  givenDate         DateTime
endVar


fullDays[1] = "Sunday"
fullDays[2] = "Monday"
fullDays[3] = "Tuesday"
fullDays[4] = "Wednesday"
fullDays[5] = "Thursday"
fullDays[6] = "Friday"
fullDays[7] = "Saturday"


givenDate = DateTime("5:35:20 AM 12/25/93")
  ; this displays "Saturday" in a dialog box
msgInfo("Day of the week", fullDays[dowOrd(givenDate)])


endMethod
```

doy method

Returns the number representing a specified day's position in the year.

Syntax

```
doy ( ) SmallInt
```

Description

doy returns an integer from 1 to 366 representing a specified day's position in the year. January 1 is day one, February 1 is day 32, and so on. If the DateTime value given is invalid, the method fails.

Example

{button ,AL(`OPAL_TYPE_DATETIME;OPAL_METH_DTDOW;OPAL_METH_DTMOY;',0,"Defaultoverview",)}

Related Topics

doy example

The following example uses a button's **pushButton** method to display a day's position in a specified year. This example assumes the current date and time format is in the form hh:mm:ss am/pm mm/dd/yy.

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
  theDate DateTime
endVar


theDate = DateTime("5:35:20 AM 6/1/92")


  ; this displays "5:35:20, 6/1/92 is
  ; 153 days past the first of the year"
msgInfo("Date", String(theDate) + " is " + String(theDate.doy()) +
                " days past the first of the year.")


endMethod
```

hour method

Extracts the hour from a specified DateTime value.

Syntax

```
hour ( ) SmallInt
```

Description

hour returns an integer representing the hour of the day in the 24-hour format. If the DateTime value given is not valid, the method fails.

Example

{button ,AL(`OPAL_TYPE_DATETIME;OPAL_METH_DTDAY;OPAL_METH_DTMONTH;OPAL_METH_DTMINUTE;OPAL_METH_DTMILLISEC;OPAL_METH_DTYEAR;',0,"Defaultoverview",)} **Related Topics**

hour example

The following example extracts the hour from a specified DateTime and displays it in a dialog box. If the DateTime value is specified in the 12-hour format, **hour** returns its 24-hour equivalent.

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
  dt DateTime
endVar


dt = DateTime("8:15:18 pm 12/29/92")
msgInfo("Hour", dt.hour())     ; displays 20 in a dialog


endMethod
```

isLeapYear method

Reports whether a year has 366 days.

Syntax

isLeapYear ( ) Logical

Description

isLeapYear returns True if the year within a specified DateTime value has 366 days; otherwise, it returns False. If the DateTime value given is not valid, the method fails.

Note

- **isLeapYear** returns True for all B.C. era dates.

Example

{button ,AL(`OPAL_TYPE_DATETIME;OPAL_METH_DTYEAR;',0,"Defaultoverview",)} Related Topics

isLeapYear example

The following example uses the **pushButton** method for the *testLeapYr* button to display True if the specified DateTime is a leap year; otherwise the method displays False. This code assumes the current date and time format is in the form hh:mm:ss am/pm mm/dd/yy.

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
  bDay     DateTime
  leapYear Logical
endVar


bDay = DateTime("5:35:20 AM 6/1/92")


leapYear = bDay.isLeapYear()
leapYear.view("bDay")          ; displays True


endMethod
```

milliSec method

Extracts the milliseconds from a DateTime.

Syntax

```
milliSec ( ) SmallInt
```

Description

milliSec returns an integer representing the milliseconds specified in a DateTime value. If the DateTime value given is not valid, the method fails.

Example

{button ,AL(`OPAL_TYPE_DATETIME;OPAL_METH_DTHOUR;OPAL_METH_DTMINUTE;OPAL_METH_DTSECOND;',0,"Defaultoverview",)} **Related Topics**

milliSec example

The following example constructs a DateTime value from integer calculations and displays the milliseconds in a dialog box.

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
  dt DateTime
  oneSecond, oneMinute, oneHour LongInt
endVar
oneSecond = 1000                ; milliseconds
oneMinute = oneSecond * 60
oneHour   = oneMinute * 60


  ; the following statement assigns dt a DateTime value
  ; of "1:20:30.4 pm 00/00/00" (the statement does not
  ; assign a date, so DateTime sets date portion to 0)
dt = DateTime(13 * oneHour   +
              20 * oneMinute +  ; specifies 1:20 pm
              30 * oneSecond +  ; + 30 seconds
              400)              ; + 400 milliseconds


msgInfo("Milliseconds", dt.milliSec())   ; displays 400


endMethod
```

minute method

Extracts the minutes from a DateTime.

Syntax

minute ( ) SmallInt

Description

minute returns an integer representing the minutes in a specified DateTime value. If the DateTime value given is not valid, the method fails.

Example

{button ,AL(`OPAL_TYPE_DATETIME;OPAL_METH_DTHOUR;OPAL_METH_DTMILLISEC;OPAL_METH_DTSECOND;',0,"Defaultoverview",)} Related Topics

minute example

The following example uses the **pushButton** method for *thisButton* to display the minutes in a specified DateTime. This code assumes the current date and time format is in the form hh:mm:ss am/pm mm/dd/yy.

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
  dt DateTime
endVar


dt = DateTime("9:20:15 am 8/2/93")


msgInfo("Minutes", dt.minute())    ; displays 20


endMethod
```

month method

Extracts as a number the month from a specified DateTime.

Syntax

```
month ( ) SmallInt
```

Description

month returns an integer representing the specified month's position in the year. January is month one, February is month two, and so on. If the DateTime value given is not valid, the method fails.

Example

{button ,AL(`OPAL_TYPE_DATETIME;OPAL_METH_DTMOY;',0,"Defaultoverview",)} **Related Topics**

month example

The following example displays the month of the year by name rather than by abbreviation or number. This code uses **month** to retrieve the appropriate subscript of a fixed array and displays the value of the array element in a dialog box. This code is attached to the **pushButton** method for the *fullMonth* button. This example assumes the current date and time format is in the form hh:mm:ss am/pm mm/dd/yy.

```
; fullMonth::pushButton
method pushButton(var eventInfo Event)
var
  fullMonth Array[12] String
  orderDate DateTime
endVar


fullMonth[1] =  "January"
fullMonth[2] =  "February"
fullMonth[3] =  "March"
fullMonth[4] =  "April"
fullMonth[5] =  "May"
fullMonth[6] =  "June"
fullMonth[7] =  "July"
fullMonth[8] =  "August"
fullMonth[9] =  "September"
fullMonth[10] = "October"
fullMonth[11] = "November"
fullMonth[12] = "December"


orderDate = DateTime("5:35:20 AM 9/18/93")


  ; this displays "September" in a dialog box
msgInfo("Order Month", fullMonth[month(orderDate)])


endMethod
```

moy method

Extracts the month from a specified DateTime as a string.

Syntax

```
moy ( ) String
```

Description

moy returns the first three letters of the name of the specified month. If the DateTime value given is not valid, the method fails.

Example

{button ,AL(`OPAL_TYPE_DATETIME;OPAL_METH_DTMONTH;',0,"Defaultoverview",)} **Related Topics**

moy example

The following example uses the **pushButton** method for *thisButton* to display the abbreviated month name of a specified DateTime. This code assumes the current date and time format is in the form hh:mm:ss am/pm mm/dd/yy.

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
  orderDate DateTime
endVar


orderDate = DateTime("2:09:00 AM 3/3/97")
msgInfo("Order date", orderDate.moy())    ; displays Mar


endMethod
```

second method

Extracts the seconds from a specified DateTime.

Syntax

```
second ( ) SmallInt
```

Description

second returns an integer representing the seconds in a DateTime. If the DateTime value given is not valid, the method fails.

Example

{button ,AL(`OPAL_TYPE_DATETIME;OPAL_METH_DTHOUR;OPAL_METH_DTMILLISEC;OPAL_METH_DTMINUTE;',0,"Defaultoverview",)} Related Topics

second example

The following example constructs a DateTime value from integer calculation and displays the seconds the DateTime in a dialog box.

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
  dt DateTime
  oneSecond, oneMinute, oneHour LongInt
endVar
oneSecond = 1000                ; milliseconds
oneMinute = oneSecond * 60
oneHour   = oneMinute * 60


  ; the following statement assigns dt a DateTime value
  ; of "1:20:30.4 pm 00/00/00" (the statement does not
  ; assign a date, so DateTime sets date portion to 0)
dt = DateTime(13 * oneHour   +
              20 * oneMinute +   ; specifies 1:20 pm
              30 * oneSecond +   ; + 30 seconds
              400)               ; + 400 milliseconds


msgInfo("Seconds", dt.second())  ; displays 30


endMethod
```

year method

Extracts the year from a specified DateTime.

Syntax

year ( ) SmallInt

Description

year returns an integer representing the year within a specified DateTime. If the DateTime value given is invalid, this method fails.

Example

{button ,AL(`OPAL_TYPE_DATETIME;OPAL_METH_DTDAY;OPAL_METH_DTISLEAPYEAR;OPAL_METH_DTMONTH;OPAL_METH_DTMOY;',0,"Defaultoverview",)} Related Topics

year example

The following example uses *yearButton*'s **pushButton** method to display the four-digit year in a specified DateTime value. This code assumes the current date and time format is in the form hh:mm:ss am/pm mm/dd/yy.

```
; yearButton::pushButton
method pushButton(var eventInfo Event)
var
  orderDate DateTime
endVar


orderDate = DateTime("2:15:24 pm 3/3/97")
msgInfo("Order date", orderDate.year())   ; displays 1997


endMethod
```

DDE type

Dynamic data exchange (DDE) is a Windows protocol that allows Corel Paradox to share data with other applications that adopt the DDE protocol. DDE methods grant you access to data created and stored in other applications. You can also use DDE methods to send commands and data to other applications.

Notes

- When you use DDE to access Corel Paradox from another application, the Corel Paradox application name is PDXWIN32.
- Corel Paradox and ObjectPAL also support OLE, another protocol for sharing data between applications. For more information, see the OLE type and to About OLE.

Methods for the DDE type

close

execute

open

setItem

Print related ObjectPAL methods and examples

close method

Closes a DDE link.

Syntax

```
close ( )
```

Description

close ends a DDE conversation by closing the link between Corel Paradox and the other application. **close** does not affect the status of the other application.

Example

{button ,AL(`OPAL_TYPE_DDE;OPAL_METH_DDOPEN;OPAL_METH_DDSETI;',0,"Defaultoverview",)}

Related Topics

close example

The following example retrieves data from a Quattro Pro for Windows worksheet and then uses the Quattro Pro for Windows macro command {FileExit} to close Quattro Pro for Windows before the **close** method is called.

```
var
   ddeVar  DDE
   Winery  AnyType
endVar


ddeVar.open("QPW", "C:\\QPW\\SAMPLES\\WINES.WB2", "$A:$C$2")


Winery = ddeVar
msginfo("First Winery", Winery)


ddeVar.execute("{FileExit}")


ddeVar.close()
```

execute method

Uses a DDE link to send a command to another application.

Syntax

```
execute ( const command String )
```

Description

execute uses a DDE link to send the string *command* to an application. The nature of *command* varies from one application to another. For example, a string that is understood by a word processing program may not be accepted by a spreadsheet application, and spreadsheets from different manufacturers may use different commands to perform similar activities.

Example

{button ,AL(`OPAL_TYPE_DDE;OPAL_METH_DDSETI;OPAL_METH_DDOPEN;OPAL_METH_DDCLOS;',0,"Defaultoverview",)} Related Topics

execute example

See the **open** example.

open method

Opens a DDE link to another application.

Syntax

```
1. open ( const server String ) Logical
2. open ( const server String, const topic String ) Logical
3. open ( const server String, const topic String, const item String ) Logical
```

Description

open creates a DDE link to another application, and instructs the application to open a document specified in *item*

This method returns True if the application is successfully opened; otherwise, it returns False. If the server application cannot open the application this method fails.

The nature of *item* varies from one application to another. For example, a string that is understood by a word processing program may not be accepted by a spreadsheet, and spreadsheets from different manufacturers may use different commands to perform similar activities.

Note

- A DDE session can only be started with a running application, a fully-registered application, or an application that resides in the known system path (e.g., an application that is within the path statement in the Autoexec.bat).

Example

{button ,AL(`OPAL_TYPE_DDE;OPAL_METH_DDCLOS;OPAL_METH_DDSETI;OPAL_METH_DDEXEC;',0,"Defaultoverview",)} Related Topics

open example

The following example uses a Corel Paradox DDE Session to launch WordPerfect 7, minimize the application and invoke the WordPerfect Import dialog box.

This example uses **getRegistryValue** to locate the path for the WordPerfect executable, and uses **execute** to launch the application.

```
Method pushButton(var eventInfo Event)
var
wpDDE   dde  ;declare a variable of DDE type
strLevel    String  ;declare a variable of string type
endVar
    strLevel = getRegistryValue( "Software\\Microsoft\\Windows\\CurrentVersion\\App Paths\\
WPWin.exe","",
    RegKeyLocalMachine ) ; check registry for path to WordPerfect application

 IF NOT execute(strLevel ) THEN     ;attempt to launch WordPerfect
 MSGINFO("Stop","Could not find WordPerfect!") ;alert user if launch failed
 else
sleep(5000)  ; sleep allows WordPerfect time to open and get ready to accept DDE commands

 if not wpDDE.open(WPWin7_MACROS","commands") then  ;attempt to start DDE dialog with
WordPerfect
 sleep (5000);sleep some more in case WordPerfect isn't fully open
 wpDDE.open("WPWin7_MACROS","commands") ; try DDE link again
 wpDDE.execute("AppMinimize()")         ; minimize WordPerfect
 wpDDE.execute("importdlg ()")          ; open WordPerfect's import dialog box
else
wpDDE.execute("AppMinimize()")          ; minimize WordPerfect
 wpDDE.execute("importdlg ()")          ;open WordPerfect's import dialog box
 endif
 endif
endMethod
```

setItem method

Specifies an item in a DDE conversation.

Syntax

```
setItem ( const item String )
```

Description

setItem specifies an item in a DDE link where the application and topic are established. The argument *item* specifies a new item. The nature of *item* varies from application to application. For example, a string that is understood by a word processing program may not be accepted by a spreadsheet, and spreadsheets from different manufacturers may use different commands to perform similar activities.

Example

{button ,AL(`OPAL_TYPE_DDE;OPAL_METH_DDEXEC;OPAL_METH_DDCLOS;;',0,"Defaultoverview",)}

Related Topics

setItem example

The following example uses **setItem** to retrieve the values of two cells in a QPW worksheet:

```
var
   winesLink            DDE
   Appellation, Region  AnyType
endVar


; link to the QPW worksheet
winesLink.open("QPW", "C:\\QPW\\SAMPLES\\WINES.WB2")


winesLink.setItem("$A:$D$2")    ;// item is cell A:D2
Appellation = winesLink         ;// sets Appellation = cell D2


winesLink.setItem("$A:$E$2")    ;// item is cell A:E2
Region = winesLink              ;// sets Region = cell E2


msgInfo("Wines Information",
        "Appellation is: " + String(Appellation)+
        ", Region is " + String(Region))


winesLink.close()
```

DynArray type

A DynArray is a flexibly structured dynamic array. Using a DynArray, you can retrieve values quickly, even when the dynamic array contains a large number of items.

This type of array is dynamic because you do not specify its size. Instead, a DynArray's dimensions automatically change as items are added or removed. A DynArray's size is limited only by system memory.

ObjectPAL also supports fixed-size and resizeable arrays. For more information, see Array type.

The indexes of dynamic arrays are not integers; dynamic array indexes (also called keys) can be any valid ObjectPAL expression that evaluates to a String. Each index in a dynamic array is associated with a value.

The following table list the methods of the DynArray type, including several derived methods from the AnyType type.

Methods for the DynArray type

AnyType	DynArray
blank	**contains**
dataType	**empty**
isAssigned	**getKeys**
isBlank	**removeItem**
isFixedType	**size**
	view

Print related ObjectPAL methods and examples

contains method

Searches the indexes in a DynArray.

Syntax

```
contains ( const value AnyType ) Logical
```

Description

contains returns True if an elements index in a DynArray matches the specified value, character for character; otherwise, it returns False. **contains** is not case sensitive.

Example

{button ,AL(`OPAL_TYPE_DYNARRAY;OPAL_METH_DYGETKEYS;OPAL_METH_ATVIEW;',0,"Defaultoverview",)} Related Topics

contains example

The following example uses **contains** to test whether a dynamic array index corresponds to a menu item. In this example, the form's **open** method creates a menu and assigns several values to a dynamic array. When the user selects an item from the menu, the form's **menuAction** method compares the menu selection with indexes in the DynArray. If a DynArray index is defined for the selected menu item, the **menuAction** method displays the value associated with that DynArray element; otherwise it displays the value of another element.

The following code goes in the form's Var window:

```
; thisForm::Var
var
  msg DynArray[] AnyType    ; stores messages
  m1             Menu       ; menu bar
  p1             PopUpMenu  ; pop-up attached to menu item
  choice         String     ; user's menu selection
endVar
```

The following code is attached to the form's **open** method:

```
; thisForm::open
method open(var eventInfo Event)
if eventInfo.isPreFilter()
  then
    ;code here executes for each object in form
  else
    ;code here executes just for form itself


    p1.addText("Time")             ; add items to the pop-up menu
    p1.addText("Date")
    p1.addText("Colors")


    m1.addPopUp("&Utilities", p1)  ; attach the pop-up to a menu bar item
    m1.show()                      ; show the menu bar


      ; Now initialize the msg dynamic array. msg Indexes correspond to
      ; the pop-up menu items generated above. msg values are values that
      ; appear in a dialog box when the user selects a menu. Note that
      ; msg does NOT contain a "Colors" index.
    msg["Time"] = time()           ; show current date for "Time" selection
    msg["Date"] = date()           ; show current date for "Date" selection
    msg["Error"] = "Sorry, this menu selection is not implemented."


endif
endMethod
```

The following code is attached to the form's **menuAction** method:

```
; thisForm::menuAction
method menuAction(var eventInfo MenuEvent)
if eventInfo.isPreFilter()
  then
    ;code here executes for each object in form


    choice = eventInfo.menuChoice()


    if isBlank(choice) = False then    ; if user selected a menu
      if msg.contains(choice) then     ; if selection matches an index in
                                       ; the msg dynamic array
        msgInfo(choice, msg[choice])   ; display the value of that element
      else                             ; else selection didn't match an element
        msgStop("Stop!", msg["Error"]) ; display the value of another element
      endif
    endif


  else
    ;code here executes just for form itself
endif
```

```
endMethod
```

empty method

Removes all items from a dynamic array.

Syntax

```
empty ( )
```

Description

empty removes all items from an dynamic array. The size of the DynArray becomes 0.

Example

{button ,AL(`OPAL_TYPE_DYNARRAY;OPAL_METH_DYCON;',0,"Defaultoverview",)} Related Topics

empty example

The following example removes all items from a dynamic array. The code immediately following declares a dynamic array in a form's Var window. This dynamic array is global to all objects on the form.

```
; thisForm::Var
Var
  myCar DynArray[] AnyType  ; declare a dynamic array
endVar
```

The following code is attached to the **pushButton** method of the *fillButton*. When this button is pressed, the code assigns several elements of the *myCar* DynArray.

```
; fillButton::pushButton
method pushButton(var eventInfo Event)


myCar["Make"]  = "Porsche"  ; load the DynArray
myCar["Model"] = "911 sc"
myCar["Color"] = "Dark Blue"
myCar["Year"]  = 1986
  ; display myCar DynArray and indicate size in the title (4)
myCar.view("myCar size: " + String(myCar.size()))
endMethod
```

The following code is attached to the **pushButton** method of the *emptyButton* button. When this button is pressed, the code empties the *myCar* array and displays its contents.

```
; emptyButton::pushButton
method pushButton(var eventInfo Event)
myCar.empty()     ; empty the myCar DynArray

  ; display myCar DynArray and indicate size in the title (0)
myCar.view("myCar size: " + String(myCar.size()))
endMethod
```

getKeys method

Loads the indexes of an existing DynArray into a resizeable array.

Syntax

```
getKeys ( var keyNames Array[ ] String )
```

Description

getKeys creates the resizeable array specified in *keyNames* and assigns the index in the DynArray to the values of each element. This method loads the index values from a DynArray into a resizeable array. If *keyNames* exists, **getKeys** overwrites it without asking for confirmation. Index values are sorted into the new array so that the lowest index value becomes *keyNames*[1].

Example

{button ,AL(`OPAL_TYPE_DYNARRAY;OPAL_METH_DYCON;',0,"Defaultoverview",)} Related Topics

getKeys example

The following example assigns several elements to the *myCar* DynArray and then uses **getKeys** to create an array that stores the *myCar* indexes. The results are displayed in a **view** dialog box.

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
  myCar DynArray[] AnyType
  ar    Array[] String
endVar


; add some elements to the DynArray
myCar["Make"]  = "Porsche"  ; load the DynArray
myCar["Model"] = "911 sc"
myCar["Color"] = "Dark Blue"
myCar["Year"]  = 1986


; now grow ar to 4 items then view the
; new array in a dialog box
myCar.getKeys(ar)
ar.view()


; displays
;  Color      (ar[1])
;  Make       (ar[2])
;  Model      (ar[3])
;  Year       (ar[4])

endMethod
```

removeItem method

Deletes a specified item from a DynArray.

Syntax

```
removeItem ( const value AnyType )
```

Description

removeItem deletes the item in *value* (specified by its index) from a DynArray. **removeItem** is not case sensitive.

Example

{button ,AL(`OPAL_TYPE_DYNARRAY;OPAL_METH_DYEMPTY;OPAL_METH_DYCON;',0,"Defaultoverview",)}

Related Topics

removeItem example

The following example concatenates two values in a dynamic array and uses **removeItem** to remove the obsolete element.

The following code is attached to a form's Var window:

```
; thisForm::Var
var
  CustInfo DynArray[] AnyType
endVar
```

The following code is attached to the **pushButton** method for the *getCustInfo* button. This code loads the dynamic array with street address information. Your application might have a custom method that loads the dynamic array from a table or from information entered by the user.

```
; getCustInfo::pushButton
method pushButton(var eventInfo Event)
  ; load the DynArray
CustInfo["Company"] = "Ultra-Fast Computers"
CustInfo["Street"]  = "1234 Able Street"
CustInfo["City"]    = "Anywhere"
CustInfo["State"]   = "Your State"
CustInfo["Zip"]     = "99444"
CustInfo["ZipExt"]  = "9344"


  ; display contents of the CustInfo Dynarray
CustInfo.view("Contents of CustInfo")
endMethod
```

In the following code, the value of the ZipExt element is concatenated to the value of the Zip element and the ZipExt element is removed from the dynamic array. The following code is attached to the **pushButton** method for the *catZipExt* button:

```
; catZipExt::pushButton
method pushButton(var eventInfo Event)
if CustInfo.contains("ZipExt") then
  CustInfo["Zip"] = CustInfo["Zip"] + "-" + CustInfo["ZipExt"]
  CustInfo.removeItem("ZipExt")       ; remove obsolete element
else
  msgInfo("Once is enough", "Zip code has been concatenated")
endif
  ; display the results
CustInfo.view("Contents of CustInfo")
endMethod
```

size method

Returns the number of elements in a DynArray.

Syntax

```
size ( ) LongInt
```

Description

size returns the number of elements in a DynArray.

Example

{button ,AL(`OPAL_TYPE_DYNARRAY;OPAL_METH_DYCON;',0,"Defaultoverview",)} Related Topics

size example

The following example uses the **pushButton** method for *thisButton* to create a dynamic array and displays its size in a dialog box:

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
  dy DynArray[] String
endVar


dy["Name"]     = "MAST"           ; load the DynArray
dy["Business"] = "Diving"
dy["Contact"]  = "Jane Doherty"


  ; this displays "dy has 3 elements"
msgInfo("dy", "dy has " + string(dy.size()) + " elements.")
endMethod
```

view method

Displays the contents of a DynArray in a dialog box.

Syntax

```
view ( [ const title String ] )
```

Description

view displays the indexes and elements of a DynArray in a modal dialog box. ObjectPAL execution suspends until the user closes this dialog box. *title* specifies the title of the dialog box. If you omit the title string, the dialog box is named DynArray. **view** sorts the DynArray on its index before displaying the dialog box.

Unlike other data types, DynArray values displayed in a **view** dialog box cannot be changed interactively. See **view** (AnyType type) for information about other data types.

Example

{button ,AL(`OPAL_TYPE_DYNARRAY;OPAL_METH_DYCON;',0,"Defaultoverview",)} Related Topics

view example

The following example uses the **pushButton** method for the *thisButton* button to create a dynamic array and displays its contents in a dialog box:

```
;thisButton::pushButton
method pushButton(var eventInfo Event)
var
  dy DynArray[] String
endVar


dy["one"] = "first"
dy["two"] = "second"
dy["three"] "third"
dy.view("This DynArray contains:")
  ; displays the following:
  ; This DynArray contains:
  ; one     first
  ; three   third
  ; two     second
endMethod
```

ErrorEvent type

The ErrorEvent type provides methods that allow you to retrieve and set information about ObjectPAL execution errors. The only built-in event method triggered by an ErrorEvent is **error**.

The following table lists the methods of the ErrorEvent type, including several derived methods from the Event type.

Methods for the ErrorEvent type

Event	**ErrorEvent**
errorCode	**reason**
getTarget	**setReason**
isFirstTime	
isPreFilter	
isTargetSelf	
setErrorCode	

{button ,AL(`ERROREV;',0,"Defaultoverview",)} Related Topics

Print related ObjectPAL methods and examples

User-defined error constants

You can define your own error constants within a specific range. Because the error constant range is subject to change in future versions of Corel Paradox, ObjectPAL provides the IdRanges constants UserError and UserErrorMax. These constants represent the minimum and maximum values accepted for user-defined error constants.

To define ThisError and ThatError as constants, set values in a Const window as follows:

```
Const
   ThisError = 1
   ThatError = 2
EndConst
```

To use one of these constants, add it to UserError:

```
method error(var eventInfo ErrorEvent)
   if eventInfo.errorCode() = UserError + ThisError then
      doSomething()
   endIf
endMethod
```

By adding your own constant, a value above the minimum is guaranteed. To keep the value under the maximum, use the value of UserErrorMax. You can check the value with a **message** statement:

```
message(UserErrorMax)
```

In Corel Paradox, the difference between UserError and UserErrorMax is 2046. This means the largest value that you can use for an error constant is UserError + 2046. The error code 0 is reserved to mean there is no error.

{button ,AL(`ERROREV;',0,"Defaultoverview",)} Related Topics

reason method

Reports the cause of an error.

Syntax

```
reason ( ) SmallInt
```

Description

reason returns an integer value to report the cause of an ErrorEvent. ObjectPAL provides the ErrorReasons constants for testing the value returned by **reason**.

Note

- Use **errorCode** to identify an error, and **reason** to identify the cause of an error.

Example

{button ,AL(`ERROREV;OPAL_METH_EESETREASON;OPAL_METH_EVERRORCODE;',0,"Defaultoverview",)}

Related Topics

reason example

The following example shows code which should be attached to a form's built-in **error** method. This code reports the error code, the reason, and the message associated with the error.

```
; thisForm::error
method error(var eventInfo ErrorEvent)
if eventInfo.isPreFilter()
  then
    ; code here executes for each object in form
    msgInfo("Error", eventInfo.errorCode())
    if eventInfo.reason() = ErrorWarning then
      msgInfo("Warning Error", errorMessage())
    else
      msgInfo("Critical Error", errorMessage())
    endif
    disableDefault
  else
    ; code here executes just for form itself

endif
endMethod
```

setReason method

Specifies a reason for generating an ErrorEvent.

Syntax

```
setReason ( const reasonId SmallInt )
```

Description

setReason specifies a reason for generating an ErrorEvent. This method takes an ErrorReasons constant as an argument.

Example

{button ,AL(`ERROREV;OPAL_METH_EEREASON;',0,"Defaultoverview",)} Related Topics

setReason example

The following example creates an ErrorEvent, sets the reason to ErrorWarning, and sends the ErrorEvent to the form.

```
; sendAnError::pushButton
method pushButton(var eventInfo Event)
var
   ev  ErrorEvent
endVar
ev.setErrorCode(1)         ; set an error code of 1
                           ; (any nonzero will do)
ev.setReason(ErrorWarning) ; set the reason to ErrorWarning
thisForm.error(ev)         ; send the error to the form
endMethod
```

Event type

The Event type is the base type from which the other event types (e.g., ActionEvent) are derived. Many of the methods listed here are also used by other event types as derived methods.

The following built-in event methods are triggered by events:

- **open**
- **close**
- **setFocus**
- **removeFocus**
- **newValue**
- **pushButton**

Methods for the Event type

errorCode
getTarget
isFirstTime
isPreFilter
isTargetSelf
reason
setErrorCode
setReason

 **Print related ObjectPAL methods and examples**

errorCode method

Reports the status of an error flag.

Syntax

errorCode ( ) SmallInt

Description

errorCode returns a nonzero error code if there is an error; otherwise, **errorCode** returns 0. To test for a specific error, use the ObjectPAL Errors constants (e.g., peDiskError) or a user-defined error constant. To create a list of the Error constants and the corresponding error messages, use **enumRTLErrors**.

Example

{button ,AL(`OPAL_TYPE_EVENT;OPAL_METH_EVSETERRORCODE;',0,"Defaultoverview",)} Related Topics

errorCode example

The following example assume that a form contains a field object, bound to the Quant field of the *Orders* table. When the field's value changes, this code executes the built-in code for this method and determines whether an error occurred.

```
; Quant::changeValue
method changeValue(var eventInfo ValueEvent)
   doDefault
   ; check the event to see if it has an error
   if eventInfo.errorCode() <> 0 then
      errorShow() ; Display the error message in a dialog box.
   endif
endMethod
```

getTarget method

Creates a handle to the target of an event.

Syntax

```
getTarget ( var target UIObject )
```

Description

getTarget returns in *target* the handle of the UIObject that was the target of the most recent event. The target does not change as the event bubbles up the containership hierarchy.

Example

{button ,AL(`OPAL_TYPE_EVENT;OPAL_TYPE_ACTIONEVENT;OPAL_METH_EVISPREFILTER;',0,"Defaultoverview",)} Related Topics

getTarget example

The following example assumes that a number of fields from the Customer table are placed on a form. As the user moves from field to field, the form's **setFocus** method identifies the target of the event, determines if the target is a field, and changes the field's color to light blue. This provides a more dramatic visual cue than the normal highlight. The field's previous color is stored in the global variable called *oldFieldColor*. When the focus is removed from the field, the form's **removeFocus** method restores the field to its original color. The previous field color is stored in a variable declared in the form's Var window.

```
; thisForm::Var
Var
  oldFieldColor LongInt     ; to store the previous color of the field
endVar
```

The following code is attached to the form's **setFocus** method:

```
; thisForm::setFocus
method setFocus(var eventInfo Event)
var
  targObj    UIObject
endVar
if eventInfo.isPreFilter()
  then
    ; code here executes for each object in form
    ; get the target
    eventInfo.getTarget(targObj)
    if targObj.Class = "Field" then  ; if it's a field, change its color
      oldFieldColor = targObj.Color  ; save old color in var global to form
      targObj.Color = LightBlue      ; highlight field on focus
    endif
  else
    ; code here executes just for form itself

endif
endMethod
```

This code is attached to the form's **removeFocus** method:

```
; thisForm::removeFocus
method removeFocus(var eventInfo Event)
var
  targObj  UIObject
endVar
if eventInfo.isPreFilter()
  then
    ; code here executes for each object in form
    ; get the target
    eventInfo.getTarget(targObj)
    if targObj.Class = "Field" then  ; if it's a field,
      targObj.Color = oldFieldColor  ; restore color from global var
    endif
  else
    ; code here executes just for form itself

endif
endMethod
```

isFirstTime method

Reports whether the form is handling an event for the first time before dispatching it.

Syntax

isFirstTime ( ) Logical

Description

isFirstTime reports whether the form is handling an event before dispatching it to the target object, or whether the event has been dispatched and has subsequently bubbled up the containership hierarchy. This method returns True if the form is handling the event for the first time; otherwise, it returns False. Use **isFirstTime** in the form's built-in event methods.

Example

{button ,AL(`OPAL_TYPE_EVENT;OPAL_TYPE_ACTIONEVENT;OPAL_METH_EVISPREFILTER;OPAL_BMETH_ABOUT;',0,"Defaultoverview",)} Related Topics

isFirstTime example

The following example uses **isFirstTime** with **isTargetSelf** to evaluate an event in a form-level method. This code replaces the default code for the form's **pushButton** method, which normally tests **isPreFilter**.

```
; thisForm::pushButton
method pushButton(var eventInfo Event)
var
  targObj   UIObject
endVar
; This example breaks out isFirstTime and isTargetSelf from isPreFilter.
; Three valid possibilities.
; Form's own event :
  ; isTargetSelf = True, isFirstTime = True


; Dispatched events (prefiltered events):
  ; isTargetSelf = False, isFirstTime = True


; Bubbled events (explicitly passed):
   ; isTargetSelf = False, isFirstTime = False


; For the form, isTargetSelf is never True when isFirstTime is False.


eventInfo.getTarget(targObj)    ; get the target to targObj
switch
  case eventInfo.isTargetSelf() AND eventInfo.isFirstTime() :
  ; This happens only when the form is handling its own event.
       msgInfo("Status",
               "This line will not execute for pushButton events.")


  case NOT eventInfo.isTargetSelf() AND eventInfo.isFirstTime() :
    ; This happens only when the form is dispatching an event
    ; for another object. isPrefilter returns True.


       msgInfo("Status", "Dispatching a pushButton event to "
                         + targObj.Name + ".")


  case NOT eventInfo.isTargetSelf() AND NOT eventInfo.isFirstTime() :
    ; The event has been explicitly bubbled back to the form.
    ; isPrefilter returns False.


       msgInfo("Status", "A pushButton Event " +
               "has been explicitly bubbled back to the form.")
endswitch


endMethod
```

The following code is attached to the **pushButton** method for the form's *testPassEvent* button. When the form's **pushButton** method has prefiltered the event and dispatched it to the button, the button's **pushButton** method returns it to the form with the **passEvent** command. When the event returns to the form, the methods **isTargetSelf**, **isFirstTime**, and **isPrefilter** return False.

```
; testPassEvent::pushButton
method pushButton(var eventInfo Event)
passEvent    ; bubble the event up the hierarchy
endMethod
```

isPreFilter method

Reports whether the form is handling an event for another object.

Syntax

```
isPreFilter ( ) Logical
```

Description

isPreFilter reports whether the form is handling an event for another object. This method returns True when the target is some object other than the form and the form has not already handled this event. **isPreFilter** is logically equivalent to the form evaluating the following statement:

```
if (NOT eventInfo.isTargetSelf()) AND eventInfo.isFirstTime()
```

This method returns True for all internal methods, and for all external methods when they first reach the form. When external methods bubble back to the form, this method returns False. See About built-in methods for information about internal and external methods.

Note

- Form methods are *not* prefiltered. When an event occurs for the form, **isPreFilter** returns False.

Example

{button ,AL(`OPAL_TYPE_EVENT;OPAL_TYPE_ACTIONEVENT;OPAL_METH_EVISFIRSTTIME;OPAL_BMETH_ABOUT;',0,"Defaultoverview",)} Related Topics

isPreFilter example

See the example for **getTarget**.

isTargetSelf method

Reports whether an object is the target of an event.

Syntax

isTargetSelf ( ) Logical

Description

isTargetSelf reports whether an object is the target of an event. Use **isTargetSelf** in the form's built-in event methods .

Example

{button ,AL(`OPAL_TYPE_EVENT;OPAL_METH_EVISFIRSTTIME;OPAL_BMETH_ABOUT;',0,"Defaultoverview" ,)} Related Topics

isTargetSelf example

See the **isFirstTime** example.

reason method

Reports why an event occurred.

Syntax

reason ( ) SmallInt

Description

reason returns an integer value to report why an event occurred. The return value depends on the event type. ObjectPAL provides the ValueReasons constants for testing the value returned by **reason**. ErrorReasons constants are defined for ErrorEvents, MenuReasons constants for MenuEvents, MoveReasons constants for MoveEvents, and StatusReasons constants for StatusEvents.

The **reason** method is valid for other event types, including ActionEvent, KeyEvent, MouseEvent, and ValueEvent, but returns a value of zero. **setReason** is also valid for ActionEvent, KeyEvent, MouseEvent, and ValueEvent, but can only be uses to set user-defined Reason constants.

Example

{button ,AL(`OPAL_TYPE_EVENT;OPAL_TYPE_ACTIONEVENT;OPAL_TYPE_MOVEEVENT;OPAL_TYPE_STATUSEVENT;OPAL_TYPE_ERROREVENT;OPAL_TYPE_VALUEEVENT;OPAL_METH_EVSREA;OPAL_BMETH_ABOUT;',0,"Defaultoverview",)} **Related Topics**

reason example

The following example assumes that a form contains a multi-record object bound to the *Orders* table, and that the *Ship_VIA* field is a set of radio buttons. Assume also that the form is in Edit mode. The **newValue** method for *Ship_VIA* displays a message indicating why **newValue** was called. When the form opens, the Reason will be StartupValue.

```
; Ship_VIA::newValue
method newValue(var eventInfo Event)
; show why the newValue method was called
msgInfo("newValue reason",
    iif(eventInfo.reason() = StartupValue, "StartupValue",
    iif(eventInfo.reason() = FieldValue, "FieldValue", "EditValue")))
endMethod
```

When the user scrolls through the table or clicks the *nextRec* button, the Reason will be FieldValue.

```
; nextRec::pushButton
method pushButton(var eventInfo Event)
action(DataNextRecord)    ; this triggers a newValue for Ship_Via
                          ; with a Reason constant FieldValue
endMethod
```

When the user chooses a different radio button on Ship_VIA or clicks the *changeRadio* button, the Reason will be EditValue.

```
; changeRadio::pushButton
method pushButton(var eventInfo Event)
ORDERS.Ship_Via = "US Mail"    ; this triggers a newValue for Ship_Via
                               ; with a Reason of EditValue
endMethod
```

setErrorCode method

Sets the error code for an event.

Syntax

```
setErrorCode ( const errorId SmallInt )
```

Description

setErrorCode sets the error code for an event packet. If *errorId* is 0, it means there has been no error, and any non zero value for *errorId* indicates an error. To indicate a specific error, use an EventErrorCodes constant or a user-defined error constant.

Calling **setErrorCode** is not the same as calling **errorLog**, which adds error information directly to the error stack. **setErrorCode** adds the error code to the current event packet. This code may be added to the error stack, depending on how custom code and built-in code handles it.

Example

{button ,AL(`OPAL_TYPE_EVENT;OPAL_TYPE_ACTIONEVENT;OPAL_METH_EVERRORCODE;OPAL_BMETH_ABOUT;',0,"Defaultoverview",)} Related Topics

setErrorCode example

See the **errorCode** example.

setReason method

Specifies a reason for generating a move.

Syntax

```
setReason ( const reasonId SmallInt )
```

Description

setReason specifies in *reasonId* a reason for generating an event in an object's built-in **newValue** method, where *reasonId* is a ValueReasons constant.

Note

- ErrorReasons constants are defined for ErrorEvents, MenuReasons constants for MenuEvents, MoveReasons constants for MoveEvents, and StatusReasons constants for StatusEvents. See the entry for **setReason** in those sections for examples. **setReason** is also valid for ActionEvent, KeyEvent, MouseEvent, and ValueEvent, but can be used only to set user-defined Reason constants.

Example

{button ,AL(`OPAL_TYPE_EVENT;OPAL_TYPE_ACTIONEVENT;OPAL_METH_EVREAS;OPAL_BMETH_ABOUT;', 0,"Defaultoverview",)} Related Topics

setReason example

The following example assumes that a form contains a multi-record object bound to the *Orders* table, and that the *Ship_VIA* field is a set of radio buttons. The **newValue** method for *Ship_VIA* displays a message indicating why **newValue** was called.

```
; Ship_VIA::newValue
method newValue(var eventInfo Event)
; show why the newValue method was called
msgInfo("newValue reason",
        iif(eventInfo.reason() = StartupValue, "StartupValue",
        iif(eventInfo.reason() = FieldValue, "FieldValue", "EditValue")))
endMethod
```

The following example demonstrates how to set a **reason** for an event and send the event to an object.

```
; triggerValReason::pushButton
method pushButton(var eventInfo Event)
var
  ev  Event
endVar
ev.setReason(FieldValue)       ; set a reason constant for the event
ORDERS.Ship_VIA.newValue(ev)   ; send the event to the Ship_VIA field
endMethod
```

FileSystem type

FileSystem variables provide access to and information about disk files, drives, and directories. They provide a handle, a variable you can use in ObjectPAL statements to work with a directory or a file. You can use **findFirst** to view available information and initialize the FileSystem variable.

Methods for the FileSystem type

accessRights
clearDirLock
copy
delete
deleteDir
drives
enumFileList
existDrive
findFirst
findNext
freeDiskSpace
fullName
getDir
getDrive
getFileAccessRights
getValidFileExtensions
isDir
isFile
isFixed
isRemote
isRemovable
isValidDir
isValidFile
makeDir
name
privDir
rename
setDir
setDirLock
setDrive
setFileAccessRights
setPrivDir
setWorkingDir
shortName
size
splitFullFileName
startUpDir
time
totalDiskSpace
windowsDir
windowsSystemDir
workingDir

Print related ObjectPAL methods and examples

accessRights method

Reports a file's access rights.

Syntax

```
accessRights ( ) String
```

Description

accessRights returns a string which describes the file's access rights. Access rights can be one or more of the following: A, D, H, R, S, V (for archive, directory, hidden, read-only, system, and volume, respectively). If **accessRights** returns an empty string, the file has no attributes set. You must use **findFirst** before using **accessRights**.

Example

{button ,AL(`OPAL_TYPE_FILESYSTEM;OPAL_METH_FSGETFILEACCESSRIGHTS;OPAL_METH_FSSETFILEACCESSRIGHTS;',0,"Defaultoverview",)} Related Topics

accessRights example

Checks the attributes of the file MEMO14.TXT. Calls **findFirst** to ensure that the file exists and then calls **accessRights**. If the file is writable, calls Notepad so you can edit the file.

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
   fileName  String
   fs        FileSystem
endVar


fileName = "c:\\Corel\\Suite8\\Paradox\\myfiles\\memo14.txt"


if fs.findFirst(fileName) then


      ; if file attributes include R (read only)
   if search(fs.accessRights(), "R") > 0 then
      msgStop(fileName, "This file is marked read-only.")
   else
      ; run notepad editor for the file
      execute("NotePad.exe " + fileName)
   endIf
else
   msgStop("Error", "Can't find " + fileName)
endIf


endMethod
```

clearDirLock procedure

Unlocks a specified directory.

Syntax

```
clearDirLock ( const dirName String ) Logical
```

Description

clearDirLock removes a directory lock from the directory specified in *dirName*. This method returns True if it succeeds; otherwise, it returns False.

Example

{button ,AL(`OPAL_TYPE_FILESYSTEM;OPAL_METH_FSSETDIRLOCK;',0,"Defaultoverview",)} Related Topics

clearDirLock example

See the **setDirLock** example.

copy method

Copies a file.

Syntax

```
copy ( const srcName String, const dstName String ) Logical
```

Description

copy returns True if successful in copying source file *srcName* to destination file *dstName*; otherwise, it returns False. If *dstName* exists, this method overwrites the file without asking for confirmation. This method copies only one file at a time and does not accept DOS wildcard characters.

Example

**{button ,AL(`OPAL_TYPE_FILESYSTEM;OPAL_METH_FSRNAM;OPAL_METH_FSDEL;',0,"Defaultoverview",)
} Related Topics**

copy example

Searches the current directory for *sourceFile*. If *sourceFile* exists, copy creates a new file called *destFile*, which contains the original file's information.

```
; copyButton::pushButton
method pushButton(var eventInfo Event)
var
   fs           FileSystem
   sourceFile,
   destFile     String
endVar


sourceFile = "memo14.txt"
destFile = "memo14.bak"


if fs.findFirst(sourceFile) then
   if fs.copy(sourceFile, destFile) then
      message(sourceFile + " copied to " + destFile)
   else
      message("Copy failed...")
   endif
else
   msgInfo(sourceFile, "File not found.")
endIf


endMethod
```

delete method

Deletes a file.

Syntax

```
delete ( const name String ) Logical
```

Description

Returns True if it deletes the specified file; otherwise, returns False. This method can delete only one file at a time and does not accept DOS wildcard characters.

Examples

{button ,AL(`OPAL_TYPE_FILESYSTEM;OPAL_METH_FSDELD;',0,"Defaultoverview",)} **Related Topics**

delete method examples

Example1 Displaying a Yes/No dialog box

Example2 Using a **while** loop to delete files

delete example 1

Displays a dialog box asking whether you want to delete *fileName*. If you choose Yes, **delete** deletes the file.

```
; delOne::pushButton
method pushButton(var eventInfo Event)
var
    fs       FileSystem
    oldFile  String
endVar


fileName = "MyText.old"


if fs.findFirst(fileName) then
   if msgYesNoCancel("Delete?", fileName) = "Yes" then
      fs.delete(fileName)
   endIf
else
   msgInfo(fileName, "File not found.")
endIf


endMethod
```

delete example 2

Uses a **while** loop to delete files with the .OLD extension in the current directory.

```
; delAll::pushButton
method pushButton(var eventInfo Event)
var
   fs  FileSystem
endVar


if fs.findFirst("*.old") then
   fs.delete(fs.name())
   while fs.findNext()
      fs.delete(fs.name())
   endWhile
else
   msgInfo("*.OLD", "File not found.")
endIf


endMethod
```

deleteDir method

Deletes a directory, but only if the directory is empty (contains no files).

Syntax

```
deleteDir ( const name String ) Logical
```

Description

Returns True if successful in deleting the specified directory; otherwise, returns False. This method does not prompt for confirmation before deleting.

Examples

{button ,AL(`OPAL_TYPE_FILESYSTEM;OPAL_METH_FSDEL;OPAL_METH_FSMDIR;',0,"Defaultoverview",)}

Related Topics

deleteDir method examples

Example1	Simple deletion
Example2	Deleting a directory by first creating an array

deleteDir example 1

Deletes the directory (folder) C:\DOS. If the C:\DOS folder contains files, deleteDir cannot delete it and an error message is displayed.

```
; delDOS::pushButton
method pushButton(var eventInfo Event)
var
   fs  FileSystem
endVar


if fs.findFirst("c:\\dos") then
   if not fs.deleteDir("c:\\dos") then
      msgStop("Error", "Could not delete directory.")
   endIf
endIf


endMethod
```

In the following code, **enumFileList** checks whether the directory C:\SCAN\SUBSCAN is empty. If so, it creates an array containing one item (the directory name), and **deleteDir** deletes the directory:

```
; delDir1::pushButton
method pushButton(var eventInfo event)
var
   fs  FileSystem
   fileNames Array[] String
endVar


fs.enumFileList("c:\\scan\\subscan", fileNames)


; compare size to 1 because directory has no filespec
if fileNames.size() = 1 then
   fs.deleteDir("c:\\scan\\subscan")
else
   msgStop("Stop", "Directory is not empty.")
endIf


endMethod
```

deleteDir example 2

deleteDir deletes the directory (folder) C:\SCAN\SUBSCAN. Before the directory is deleted, **enumFileList** creates an array that contains the current directory and its parent directory.

```
; delDir2::pushButton
method pushButton(var eventInfo event)
var
   fs  FileSystem
   fileNames Array[] String
endVar


fs.enumFileList("c:\\scan\\subscan\\*.*", fileNames)


; compare size to 2 because directory has the *.* filespec
if fileNames.size() = 2 then ; size = 2 because of *.* filespec
   fs.deleteDir("c:\\scan\\subscan")
else
   msgStop("Stop", "Directory is not empty.")
endIf


endMethod
```

drives method

Returns the letters of the drives attached to the system and known to Windows.

Syntax

drives ( ) String

Description

drives returns a string containing the letters of the drives that are attached to the system and known to Windows.

Example

{button ,AL(`OPAL_TYPE_FILESYSTEM;OPAL_METH_FSEXDR;',0,"Defaultoverview",)} **Related Topics**

drives example

Displays a dialog box listing the ID letters of the drives that are attached to the system:

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
   fs  FileSystem
endVar


; this displays a list of attached drives
; example:  ABCHJKXY
msgInfo("Drives", fs.drives())


endMethod
```

enumFileList method

Lists information about files.

Syntax

```
1. enumFileList ( const fileSpec String, var arrayName Array[ ] String )
2. enumFileList ( const fileSpec String, const tableName String )
```

Description

enumFileList lists information about files that match the criteria specified in *fileSpec*. If *fileSpec* is *.*, the array or table includes records for the current directory (.) and the parent directory (..).

Syntax 1 writes data to the array *arrayName,* which you must declare before calling this method. The resulting array contains filenames and extensions, but does not contain paths.

Syntax 2 writes data to the table *tableName*. If the table does not exist, creates it automatically and enumerates the file list. If *tableName* does not specify a path, enumFileList creates the table in the working directory. If the table exists and is open, this method appends data to it; if the table is closed, overwrites its data.

The following table describes the structure of the table:

Field name	Type & size		Description
Name	Alpha	255	Filename (and extension
Size	Numeric		File size in bytes
Attributes	Alpha	10	DOS file attributes
Date	Alpha	10	Date of last modification
Time	Alpha	10	Time of last modification

enumFileList lists filenames in the same order as the directory.

Example

{button ,AL(`OPAL_TYPE_FILESYSTEM;OPAL_METH_FSVALFE;OPAL_METH_FSISFIL;',0,"Defaultoverview",)} Related Topics

enumFileList example

Demonstrates both syntaxes of **enumFileList**. First, **enumFileList** searches the specified directory (folder) for forms and uses Syntax 1 to create an array of filenames, which is displayed in a pop-up menu. Then, **enumFileList** uses Syntax 2 to create a table of information about the files in a Table window.

```
; demoButton::pushButton
method pushButton(var eventInfo Event)
var
   fs  FileSystem
   formDir,  theForm String
   formNames Array[] String
   tv  tableView
   p  PopUpMenu
endVar


formDir = "C:\\Corel\\Suite8\\Paradox\\samples\\*.f?l"


if fs.findFirst(formDir) then          ; if one *.f?l is found
   fs.enumFileList(formDir, formNames)  ; create an array of *.f?l files
   p.addArray(formNames)                ; show the array in a pop-up menu
   theForm = p.show()                   ; display a pop-up menu of filenames
endIf


if fs.findFirst(formDir) then          ; if one *.f?l is found
   fs.enumFileList(formDir, "forms.db") ; create FORMS.DB listing *.f?l files
   tv.open("forms.db")                  ; display FORMS.DB table
endIf


endMethod
```

existDrive method

Reports whether a drive is attached to the system.

Syntax

```
existDrive ( const driveLetter String ) Logical
```

Description

existDrive returns True if the specified drive is attached to the system; otherwise, it returns False. You can specify the drive using a letter (C) or a letter and a colon (C:).

Example

{button ,AL(`OPAL_TYPE_FILESYSTEM;OPAL_METH_FSDRIV;OPAL_METH_FSGDRV;OPAL_METH_FSIRMO;OPAL_METH_FSIRMV;',0,"Defaultoverview",)} Related Topics

existDrive example

Calls **existDrive** to check whether drive P exists. If **existDrive** returns True, **setDrive** sets drive P as the default drive.

```
; checkDrive::pushButton
method pushButton(var eventInfo Event)
var
   fs         FileSystem
   driveName  String
endVar


driveName = "P"


if fs.existDrive(driveName) then
   fs.setDrive(driveName)
else
   msgStop("Stop", "Drive " + driveName + " is not attached.")
endIf


endMethod
```

findFirst method

Searches a file system for a filename.

Syntax

```
findFirst ( const pattern String ) Logical
```

Description

findFirst returns True if a file is found whose name matches *pattern*; otherwise, it returns False. *pattern* may contain the DOS wildcard characters * and ?, as used with the DOS command DIR. Examples of pattern include:

- C:*.*
- ..\\myDir*.*
- *.txt
- fr*.db?

Use **findFirst** to check whether a file or directory exists and to initialize a FileSystem variable before calling another FileSystem method or procedure. You must fully qualify **findFirst** calls to other than the current default drive or path, unless you reset the default drive and path with the **setDir** method.

Under Windows 95, **findFirst** also finds the 8.3 format of the filename that exists in the file system for long filenames.

Note

- **findFirst** finds file and directory names in the order that they're listed in the directory. The first value returned by **findFirst** depends on the path and file specification.

Example

**{button ,AL(`OPAL_TYPE_FILESYSTEM;OPAL_METH_FSFINX;OPAL_METH_FSNAME;',0,"Defaultoverview",)
} Related Topics**

findFirst example

The following example demonstrates how **findFirst** behaves depending on the file specification in *pattern*:

```
; buttonOne::pushButton
method pushButton(var eventInfo Event)
var
   fs FileSystem
endVar


; Search in the root folder for a file
; or folder named COREL\SUITE8\PARADOX.
if fs.findFirst("c:\\Corel\\Suite8\\Paradox") then
   ; this displays COREL\SUITE8\PARADOX (findFirst finds the folder)
   msgInfo("Pattern: c:\\Corel\\Suite8\\Paradox", "Name: " + fs.name())
else
   errorShow()
endIf


; >>INVALID PATTERN CAUSES AN ERROR!! <<
if fs.findFirst("c:\\Corel\\Suite8\\Paradox\\") then
   message("This message never displays.")
else
   errorShow("Invalid pattern: c:\\Corel\\Suite8\\Paradox\\")
endIf


; Search in the COREL\SUITE8\PARADOX folder for
; any file or folder.
if fs.findFirst("c:\\Corel\\Suite8\\Paradox\\*.*") then
   ; This displays one dot (.) because the
   ; first file in a directory is a single dot (.).
   msgInfo("Pattern: c:\\Corel\\Suite8\\Paradox\\*.*", "Name: " + fs.name())
else
   errorShow()
endIf


endmethod
```

findNext method

Searches a file system for multiple instances of a filename.

Syntax

```
findNext ( [ const fileSpec String ] ) Logical
```

Description

After **findFirst** succeeds, **findNext** searches for the next file whose name matches *Pattern*. **findNext** returns True if successful; otherwise, it returns False.

You can also use the optional argument *fileSpec* to specify a path and file specification. If you do, the call to **findFirst** is unnecessary.

Examples

{button ,AL(`OPAL_TYPE_FILESYSTEM;OPAL_METH_FSFIFI;',0,"Defaultoverview",)} Related Topics

findNext method examples

findNext example 1

The following example calls **findNext** to fill a list with the names of the tables in the current directory (folder). The example assumes that a field displayed as a drop-down list has already been placed in the form. The code is attached to the built-in **open** method of the list object contained by the field object.

```
; tablesFld.listObj::open
method open(var eventInfo Event)
var
   fs FileSystem
endVar


doDefault


;   This while loop fills the list in the drop-down edit
; box with *.db files in the default sample directory
while fs.findNext("c:\\Corel\\Suite8\\Paradox\\samples\\*.db")
   self.list.selection =
   self.list.selection + 1
   self.list.value = fs.name()
endWhile
endMethod
```

findNext example 2

The following example uses **findNext** with a file specification as an argument and displays a pop-up menu listing the files in the C:\COREL\SUITE8\PARADOX directory (folder):

```
; editText::pushButton
method pushButton(var eventInfo Event)
var
   fs      FileSystem
   p       PopUpMenu
   choice  String
endVar


; search for *.txt files in the COREL\SUITE8\PARADOX directory
; then add their names to a pop-up menu
while fs.findNext("c:\\Corel\\Suite8\\Paradox\\*.txt")
   p.addText(fs.name())
endWhile


choice = p.show()                   ; show the pop-up menu
if not choice.isBlank() then        ; if user selected a file
   execute("Notepad.exe " + choice)  ; edit the file in Notepad
endif


endMethod
```

freeDiskSpace method

Returns the amount of free space on a drive, measured in bytes.

Syntax

```
freeDiskSpace ( const driveLetter String ) LongInt
```

Description

freeDiskSpace returns the number of bytes available on a specified drive. You can specify the drive using a letter (C) or a letter and a colon (C:).

Examples

{button ,AL(`OPAL_TYPE_FILESYSTEM;OPAL_METH_FSTDSP;OPAL_METH_FSFIFI;OPAL_METH_FSGDRV;',0, "Defaultoverview",)} Related Topics

freeDiskSpace method examples

freeDiskSpace example 1

The following example displays a dialog box listing the number of bytes available on drive C.

```
; showCSpace::pushButton
method pushButton(var eventInfo Event)
var
   fs  FileSystem
endVar


msgInfo("Free bytes on drive C:", fs.freeDiskSpace("C"))


endMethod
```

freeDiskSpace example 2

The following example compares the size of the file MEMO14.TXT with the amount of space available on the current drive. If there's enough space, the code calls **copy** to copy the file.

```
; copyFile::pushButton
method pushButton(var eventInfo Event)
   var
      fs           FileSystem
      stDrive      String
      liFileSize,
      liFreeSpace  LongInt
      dyFileInfo   DynArray[] String
   endVar


   if fs.findFirst(":WORK:memo14.txt") then
      liFileSize = fs.size()
      splitFullFileName(workingDir(), dyFileInfo)
      stDrive = dyFileInfo["DRIVE"]
      liFreeSpace = fs.freeDiskSpace(stDrive)
   else
      msgStop("MEMO14.TXT", "File not found.")
      return
   endIf


   if liFreeSpace > liFileSize then
      fs.copy("memo14.txt", "memo14.bak")
      message("File copied successfully.")
   else
      msgStop("Copy", "Not enough disk space to copy file.")
   endIf


endMethod
```

fullName method/procedure

Returns the full path to a file.

Syntax

```
1. (Method) fullName ( ) String
2. (Procedure) fullName ( const fileName String ) String
```

Description

In Syntax 1, after a successful **findFirst** or **findNext**, **fullName** returns the full path of the found file. Use this method with **splitFullFileName** to analyze the components of a filename.

Syntax 2 operates on a filename, expanding or translating aliases and returning the expanded string. For example, if the working directory (:WORK:) is defined as C:\COREL\SUITE8\PARADOX\FORMS. Given the string :WORK:myForm.fsl Syntax 2 returns C:\COREL\SUITE8\PARADOX\FORMS\myForm.fsl.

Example

{button ,AL(`OPAL_TYPE_FILESYSTEM;OPAL_METH_FSNAME;OPAL_METH_FSFIFI;OPAL_METH_FSFINX;OPAL_METH_FSSPLIT;',0,"Defaultoverview",)} Related Topics

fullName example

The following example calls **fullName** to get the full name of the first form listed in the current directory. The code then calls **splitFullFileName** to split the name into its component parts and store them in a dynamic array. Finally, the code calls **view** to display the dynamic array.

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
   fs  FileSystem
   splitName DynArray[] String
   fullFileName  String
endVar


; if the customer.db file is in the sample directory
if fs.findFirst("c:\\Corel\\Suite8\\Paradox\\samples\\customer.db") then


   ; store the full filename to a variable
   fullFileName = fs.fullName()


   ; split filename into parts and store them in a DynArray
   splitFullFileName(fullFileName, splitName)


   ; display the component parts
   splitName.view("Split name")
endIf


endMethod
```

getDir method

Returns the path to which the FileSystem variable points.

Syntax

```
getDir ( ) String
```

Description

getDir returns a string that represents the path to which the FileSystem variable points. You can use **setDir** to make a FileSystem variable point to a specified directory. To get a drive letter, use **getDrive**.

Example

**{button ,AL(`OPAL_TYPE_FILESYSTEM;OPAL_METH_FSSDIR;OPAL_METH_FSGDRV;',0,"Defaultoverview",)
} Related Topics**

getDir example

The following example gets the path of the directory to which the FileSystem variable points, and compares it with a path. If the directories don't match, **getDir** calls **setDir** to change the directory.

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
   fs  FileSystem
   st  String
endVar


   st = "c:\\Corel\\Suite8\\Paradox\\myforms"


   if fs.getDir() <> st then
      fs.setDir(st)
   endIf
endMethod
```

getDrive method

Returns the drive letter or alias that the FileSystem variable points to.

Syntax

```
getDrive ( ) String
```

Description

getDrive returns a string representing the drive letter or alias that the FileSystem variable points to.

Example

{button ,AL(`OPAL_TYPE_FILESYSTEM;OPAL_METH_FSEXDR;OPAL_METH_FSGDIR;OPAL_METH_FSSDRV;', 0,"Defaultoverview",)} Related Topics

getDrive example

The following example calls **getDrive** to return the alias of the working directory. The code then sets the default drive to H and calls **getDrive** again to confirm the change.

```
; setH::pushButton
method pushButton(var eventInfo Event)
var
   fs        FileSystem
   newDrive  String
endVar


msgInfo("Default drive", fs.getDrive())       ; Displays :WORK:


newDrive = "H"


if fs.existDrive(newDrive) then
   if fs.setDrive(newDrive) then
      msgInfo("Default drive", fs.getDrive())   ; Displays H:
   else
      msgStop(newDrive, "Could not set drive.")
   endIf
else
   msgStop(newDrive, "Drive is not attached.")
endIf


endMethod
```

getFileAccessRights procedure

Reports a file's access rights.

Syntax

```
getFileAccessRights ( const fileName String ) String
```

Description

getFileAccessRights returns a string that describes the access rights of a file. The return values can be one or more of the following: A, D, H, R, S, V (for archive, directory, hidden, read-only, system, and volume, respectively). If **getFileAccessRights** returns an empty string, the file has no attributes set.

Example

{button ,AL(`OPAL_TYPE_FILESYSTEM;OPAL_METH_FSACCR;OPAL_METH_FSSETFILEACCESSRIGHTS;',0,"Defaultoverview",)} Related Topics

getFileAccessRights example

The following example displays the file attributes for C:\CONFIG.SYS.

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
   fileName String
endVar


fileName = "C:\\CONFIG.SYS"


msgInfo(fileName, getFileAccessRights(fileName))


endMethod
```

getValidFileExtensions procedure

Returns the valid file extensions for a specified object.

Syntax

```
getValidFileExtensions ( const objectType String ) String
```

Description

getValidFileExtensions returns a string containing the valid file extensions for the object specified in **objectType**, which is a Form, Library, Report, or Script.

Example

{button ,AL(`OPAL_TYPE_FILESYSTEM;OPAL_METH_FSGDIR;OPAL_METH_FSGDRV;OPAL_METH_FSSPLIT;', 0,"Defaultoverview",)} Related Topics

getValidFileExtensions example

The following example displays a dialog box listing the valid file extensions for forms.

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
   fx String
endVar


fx = getValidFileExtensions("Form")
msgInfo("Form file extensions:", fx)    ; displays fsl fdl


endMethod
```

isDir procedure

Reports whether a specified string represents the name of a directory.

Syntax

```
isDir ( const dirName String ) Logical
```

Description

isDir returns True if *dirName* is a valid directory name; otherwise, it returns False.

Example

{button ,AL(`OPAL_TYPE_FILESYSTEM;OPAL_METH_FSDELD;OPAL_METH_FSGDIR;OPAL_METH_FSMDIR;OPAL_METH_FSSDIR;',0,"Defaultoverview",)} Related Topics

isDir example

The following example calls **isDir** to ensure that the directory (folder) specified by the variable *newDir* is valid. If so, the code calls **setDir to** make *newDir* the default directory. In this example, the value of *newDir* is hard coded, but it can also be supplied by the user, read from a table, or extracted from another source.

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
   fs      FileSystem
   newDir  String
endVar


newDir = "C:\\Corel\\Suite8\\Paradox\\diveplan"
if isDir(newDir) then
   fs.setDir(newDir)
   msgInfo("Current directory", fs.getDir())
else
   msgStop(newDir, "Directory does not exist.")
endIf


endMethod
```

isFile procedure

Reports whether a specified string is a filename in the active file system.

Syntax

```
isFile ( const fileName String ) Logical
```

Description

isFile returns True if *fileName* is a file in the current file system; otherwise, it returns False.

Examples

{button ,AL(`OPAL_TYPE_FILESYSTEM;OPAL_METH_FSFIFI;OPAL_METH_FSISDIR;',0,"Defaultoverview",)}

Related Topics

isFile procedure examples

isFile example 1

The following example calls **isFile** and displays messages reporting whether the file specifications represent actual files.

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
   fs FileSystem
endVar


message(isFile("c:\\dos\\chkdsk.exe")) ; displays True
sleep(1500)
message(isFile("c:\\dos\\MyXFilex.ext"))   ; displays False
sleep(1500)


endMethod
```

isFile example 2

The following example asks for the full path and filename of a file to delete. The code calls **isFile** to test whether the file exists, and then calls **delete** to delete it.

```
; buttonOne::pushButton
method pushButton(var eventInfo Event)
var
   fs        FileSystem
   fileName  String
endVar


fileName = "Enter full path and filename here."
fileName.view("Delete a file")


if isFile(fileName) then         ; if the specified file exists
   fs.delete(fileName)            ; delete the file
   message("File deleted.")
else
   msgStop(fileName, "File not found.")
endIf

endMethod
```

isFixed method

Reports whether a drive is fixed (not removable or networked).

Syntax

```
isFixed ( const driveLetter String ) Logical
```

Description

isFixed returns True if the specified drive represents a fixed drive; otherwise, it returns False. You can specify the drive using a letter (C) or a letter and a colon (C:).

Example

**{button ,AL(`OPAL_TYPE_FILESYSTEM;OPAL_METH_FSIRMO;OPAL_METH_FSIRMV;',0,"Defaultoverview",)
} Related Topics**

isFixed example

In the following example, drive C is the user's local hard disk, and drive H is a network drive:

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
   fs  FileSystem
endVar


msgInfo("Is drive C fixed?", fs.isFixed("C"))  ; displays True
msgInfo("Is drive H fixed?", fs.isFixed("H"))  ; displays False


endMethod
```

isRemote method

Reports whether a drive is a remote (network) drive.

Syntax

```
isRemote ( const driveLetter String ) Logical
```

description

isRemote returns True if the specified drive represents a remote (network) drive; otherwise, it returns False. You can specify the drive using a letter (C) or a letter and a colon (C:).

Example

{button ,AL(`OPAL_TYPE_FILESYSTEM;OPAL_METH_FSIFIX;OPAL_METH_FSIRMV;',0,"Defaultoverview",)}

Related Topics

isRemote example

The following example calls **existDrive** to ensure drive H is attached and then calls **isRemote** to determine whether drive H is a network drive.

```
var
   h FileSystem
endVar
if h.existDrive("h") then ; if drive H is attached
   if h.isRemote("h") then
      msgInfo("Drive H: ", "Remote Drive")
   else
      msgInfo("Drive H:", "Not a Remote Drive.")
   endIf
else
   msgStop("Drive H", "Drive is not attached.")
endIf
```

isRemovable method

Reports whether a drive is removable.

Syntax

```
isRemovable ( const driveLetter String ) Logical
```

Description

isRemovable returns True if the specified drive is a removable drive; otherwise, it returns False. You can specify the drive using a letter (C) or a letter and a colon (C:).

Example

{button ,AL(`OPAL_TYPE_FILESYSTEM;OPAL_METH_FSIFIX;OPAL_METH_FSIRMO;',0,"Defaultoverview",)}

Related Topics

isRemovable example

The following example calls **existDrive** to ensure drive D is attached, then calls **isRemovable** to determine whether drive D is a removable drive.

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
   fs  FileSystem
   s   String
endVar


if fs.existDrive("D:") then ; if drive D is attached
   if fs.isRemovable("D") then
      msgInfo("Drive D: ", "Removable Drive")
   else
      msgInfo("Drive D:", "Not a Removable Drive.")
   endIf
endIf


endMethod
```

isValidDir procedure

Checks whether a directory name is valid.

Syntax

```
isValidDir ( const dirName String ) Logical
```

Description

isValidDir checks whether the directory name is valid for the file system. Use **isValidDir** to see if long filenames are supported on a specific volume. This procedure returns True if the directory is valid; otherwise, it returns False.

Use the **isValidFile** method to check the validity of the entire path.

Example

{button ,AL(`OPAL_TYPE_FILESYSTEM;OPAL_METH_FSISVALIDFILE;',0,"Defaultoverview",)} Related Topics

isValidDir example

See the **isValidFile** example.

isValidFile procedure

Checks whether a filename is valid.

Syntax

```
isValidFile ( const fileName String ) Logical
```

Description

isValidFile checks whether the filename is valid for the file system. Use **isValidFile** to see if long filenames are supported on a specific volume. This procedure returns True if the file is valid; otherwise it returns false.

Example

{button ,AL(`OPAL_TYPE_FILESYSTEM;OPAL_METH_FSPRIV;OPAL_METH_FSWDIR;OPAL_METH_FSWSDI;',0,"Defaultoverview",)} **Related Topics**

isValidFile example

The following example uses the view dialog to request a new filename. **isValidFile** is used to check whether the file is valid for the volume so that it can be copied to that volume.

```
proc copyNewFile( origFileName  String )
var
   newFile string
endVar


newFile.view()


if isValidFile( newFile ) then
   copy( origFileName, newFile )
else
   msgInfo( "Error", "This is not a valid filename" )
endif


endProc
```

makeDir method

Creates a new directory.

Syntax

```
makeDir ( const name String ) Logical
```

Description

makeDir creates all directories and subdirectories specified in *name*. **MakeDir** returns True if successful in creating *name* (or if the directory already exists); otherwise, it returns False.

Example

**{button ,AL(`OPAL_TYPE_FILESYSTEM;OPAL_METH_FSDELD;OPAL_METH_FSISDIR;',0,"Defaultoverview",)
} Related Topics**

makeDir example

The following example tries to create a new directory (folder) on drive C, and displays a dialog box to report success or failure.

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
   fs            FileSystem
   returnValue   Logical
endVar


; this creates \New and \New\Directory etc...
returnValue = fs.makeDir("C:\\New\\Directory\\Tree")


msgInfo("Status", iif(returnValue, "New directory created", "makeDir Failed"))


endMethod
```

name method

Returns a filename.

Syntax

name ( ) String

Description

After a successful **findFirst** or **findNext**, **name** returns the filename that matches the pattern.

Example

{button ,AL(`OPAL_TYPE_FILESYSTEM;OPAL_METH_FSFUNM;OPAL_METH_FSFIFI;OPAL_METH_FSFINX;',0, "Defaultoverview",)} Related Topics

name example

The following example calls **findFirst** and **findNext** to find the tables in the current directory and then calls **name** to create a pop-up menu listing the filenames.

```
; showName::pushButton
method pushButton(var eventInfo Event)
var
   fs  FileSystem
   p  PopUpMenu
   tv  TableView
   choice, path  String
endVar


if fs.findFirst("*.db") then    ; if a *.db file exists
   p.addStaticText("Tables")     ; create a pop-up menu
   p.addSeparator()
   p.addText(fs.name())          ; use filenames in pop-up
   while fs.findNext()
      p.addText(fs.name())
   endWhile
   choice = p.show()             ; show the menu
   if not choice.isBlank() then  ; if user selected a table
      tv.open(choice)             ; display the selected table
   endif
endIf


endMethod
```

privDir procedure

Returns the name of the private directory.

Syntax

```
privDir ( ) String
```

Description

privDir returns a string containing the full DOS path (including the drive letter) of the private directory.

Each user must have a private directory that stores temporary tables. The private directory can be on a network or on a local drive. Use **setPrivDir** to set the path to the private directory.

Example

{button ,AL(`OPAL_TYPE_FILESYSTEM;OPAL_METH_FSGDIR;OPAL_METH_FSSETPRIVDIR;OPAL_METH_FSSTART;OPAL_METH_FSWORKINGDIR;',0,"Defaultoverview",)} **Related Topics**

privDir example

The following example calls **privDir** to display the path to the private directory (:PRIV:) in the Status Bar.

```
method pushButton(var eventInfo Event)
   message("Your private directory is: ", privDir())
endMethod
```

rename method

Renames a file.

Syntax

```
rename ( const oldName String, const newName String ) Logical
```

Description

rename changes the name of the file *oldName* to *newName*. If *newName* is used by another file, the method does not overwrite the existing file. The **rename** method returns True if successful; otherwise, it returns False. **rename** is independent of **findFirst** and **findLast**.

Example

{button ,AL(`OPAL_TYPE_FILESYSTEM;OPAL_METH_FSCOPY;OPAL_METH_FSDEL;OPAL_METH_FSFIFI;OPAL_METH_FSFINX;',0,"Defaultoverview",)} Related Topics

rename example

The following example searches the current directory for the file specified in the *oldName* variable. If the file exists, the example calls **rename** to rename it. A dialog box reports any errors.

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
   fs  FileSystem
   oldName, newName  String
endVar


oldName = "memo14.txt"
newName = "memo14.bak"


if fs.findFirst(oldName) then
   if not fs.rename(oldName, newName) then
      msgStop("Could not rename file", newName + " already exists.")
   endIf
else
   msgStop(oldName, "File not found.")
endIf


endMethod
```

setDir method

Sets the directory path for a FileSystem variable.

Syntax

```
setDir ( const name String ) Logical
```

Description

setDir sets the path to *name* for a FileSystem variable. Use **setDrive** to set the default drive.

Example

**{button ,AL(`OPAL_TYPE_FILESYSTEM;OPAL_METH_FSGDIR;OPAL_METH_FSSDRV;',0,"Defaultoverview",)
} Related Topics**

setDir example

The following example calls **isDir** to check whether the directory *newDir* is valid. If the directory is valid, the code calls **setDir** to set *newDir* as the default directory.

```
method pushButton(var eventInfo Event)
var
   fs      FileSystem
   newDir  String
endVar


   newDir = "c:\\Corel\\Suite8\\Paradox\\mine\\zap"

   if isDir(newDir) then
      fs.setDir(newDir)
   else
      msgStop(newDir, "Not a valid directory.")
   endIf

   message(fs.getDir()) ; displays \Corel\Suite8\Paradox\mine\zap
endMethod
```

setDirLock procedure

Locks a specified directory.

Syntax

```
setDirLock ( const dirName String ) Logical
```

Description

setDirLock locks the directory *dirName*. The code returns True if successful; otherwise, it returns False.

A directory lock makes the directory read-only. This prevents Corel Paradox from reading from or writing to a lock file in that directory. A directory lock is required for Corel Paradox to access data from a CD-ROM drive, and can improve performance on network drives and local drives. A lock is not be respected on a local drive if Local Share is turned off.

Example

{button ,AL(`OPAL_TYPE_FILESYSTEM;OPAL_METH_FSCLEARDIRLOCK;',0,"Defaultoverview",)} Related Topics

setDirLock example

The following example calls **setDirLock** to make a network drive read-only when the form opens, and calls **clearDirLock** to remove the lock when the form closes.

The following code is attached to the form's built-in **open** method:

```
method open(var eventInfo Event)
   var
      h FileSystem
   endVar


   if eventInfo.isPreFilter() then
      ;// This code executes for each object on the form:

   else
      ;// This code executes only for the form:
      if h.existDrive("h") then ; if drive H is attached
         if h.isRemote("h") then
            setDirLock("h")
            message("Drive H: locked.")
         else
            msgStop("Drive H:", "Not a Remote Drive.")
            return
         endIf
      else
         msgStop("Drive H:", "Drive is not attached.")
         return
      endIf


   endIf
endMethod
```

The following code is attached to the form's built-in **close** method:

```
method close(var eventInfo Event)


   var
      h FileSystem
   endVar


   if eventInfo.isPreFilter() then
      ;// This code executes for each object on the form:

   else
      ;// This code executes only for the form:
      if h.existDrive("h") then ; if drive H is attached
         if h.isRemote("h") then
            clearDirLock("h")
            message("Drive H: unlocked.")
         else
            msgStop("Drive H:", "Not a Remote Drive.")
            return
         endIf
      else
         msgStop("Drive H:", "Drive is not attached.")
         return
      endIf


   endIf


endMethod
```

setDrive method

Sets a specified drive as the default drive.

Syntax

```
setDrive ( const name String ) Logical
```

Description

setDrive sets the specified drive as the default. The method returns True if successful; otherwise, it returns False. You can specify the drive with a letter (C), a letter and a colon (C:), or an alias (e.g., :MAST:).

Examples

**{button ,AL(`OPAL_TYPE_FILESYSTEM;OPAL_METH_FSGDRV;OPAL_METH_FSSDIR;',0,"Defaultoverview",)
} Related Topics**

setDrive method examples

Example1 Using **view**

Example2 Using an alias

setDrive example 1

The following example calls **view**, cast for the String type, to display a dialog box and ask for input. If you type a valid drive letter, the code calls **setDrive to** set the specified drive as the default.

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
   fs        FileSystem
   newDrive  String
endVar


newDrive = "Enter drive ID or alias here."
newDrive.view("Change default drive.") ; prompt user for input


if fs.existDrive(newDrive) then
   fs.setDrive(newDrive)
else
   msgStop(newDrive, "Drive not available.")
endIf


endMethod
```

setDrive example 2

Shows how to use an alias with **setDrive**. This example assumes that the alias (:MAST:) has already been defined.

```
; setDrive::pushButton
method pushButton(var eventInfo Event)
var
   fs FileSystem
endVar


fs.setDrive(":MAST:")


endMethod
```

setFileAccessRights procedure

Sets a file's access rights.

Syntax

```
setFileAccessRights ( const fileName String, const rights String ) Logical
```

Description

setFileAccessRights sets the access rights of a specified file to those specified in *rights. Rights* is a string that contains one or more of the following: A, D, H, R, S, V (for archive, directory, hidden, read-only, system, and volume, respectively). If *rights* is an empty string (""), **setFileAccessRights** removes all access rights settings for the specified file. You don't have to declare a FileSystem variable (or use the **findFirst** method) before calling **setFileAccessRights**.

Example

{button ,AL(`OPAL_TYPE_FILESYSTEM;OPAL_METH_FSACCR;OPAL_METH_FSGETFILEACCESSRIGHTS;',0,"Defaultoverview",)} Related Topics

setFileAccessRights example

The following example sets the file access rights for C:\CONFIG.SYS to read-only (R) and hidden (H).

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
   fileName  String
endVar


fileName = "C:\\CONFIG.SYS"


; set file attribute for CONFIG.SYS to read only and hidden
if setFileAccessRights(fileName, "RH") then
   ; if successful, display a message with the current attributes
   message (fileName + " attributes set to " +
            getFileAccessRights(fileName))
else
   ; otherwise, the procedure failed
   message("Can't set file attributes for " + fileName)
endif


endMethod
```

setPrivDir procedure

Sets or changes the private directory.

Syntax

```
setPrivDir ( const path String ) Logical
```

Description

setPrivDir sets a path to the current private directory. **setPrivDir** returns True if successful; otherwise, it returns False. The following table displays valid path values.

Value of *path*	Example
Directory name	ORDERS
Full path	C:\\COREL\\SUITE8\\PARADOX\\APPS\\ORDERS\\
Relative path	..\\..\\ORDERS
Alias	:ORDERS:

Corel Paradox closes all of its open windows and frees all locks before setting the private directory. Therefore, **setPrivDir** does not take effect until all ObjectPAL code has finished executing. You can keep a form open by adding code to its built-in **menuAction** method to trap for the MenuChangingPriv menu command (see the example for details). If you do so, save any documents that need saving before changing the working directory. **setPrivDir** returns True if successful; otherwise, it returns False.

ObjectPAL provides the following MenuCommands constants for handling changes to the private directory:

Constant	Description
MenuFilePrivateDir	Issued when the user chooses Tools, Settings, Preferences, Database, Private Directory from the Corel Paradox menu. Trap for this constant to prevent the user from changing the private directory.
MenuChangingPriv	Issued just before the private directory changes. Trap for this constant to keep a form open when changing the private directory.
MenuChangedPriv	Issued just after the private directory changes. Trap for this constant to find out when the private directory has changed.

ObjectPAL also provides the constant MenuFileWorkingDir, issued when the user clicks Tools, Settings, Preferences, Database, Working Directory.

Examples

{button ,AL(`OPAL_TYPE_FILESYSTEM;OPAL_METH_FSPRIV;OPAL_METH_FSSETWORKINGDIR;',0,"Default overview",)} Related Topics

setPrivDir procedure examples

Example1 Using a menu choice to set the private directory

Example2 Using a form's **open** and **menuAction** methods

setPrivDir example 1

The following example changes the private directory and the resulting menu commands generated by Corel Paradox. When you click Tools, Settings, Preferences, Database, Private Directory the code calls **disableDefault** to block the default behavior. This prevents Corel Paradox from displaying the Set Private Directory dialog box and then tests the value of a Logical variable *okToChangePriv* (declared and assigned elsewhere). If *okToChangePriv* is True, the code calls **setPrivDir** to set the private directory (:PRIV:) behind the scenes.

This example also handles the MenuChangingPriv menu command, issued by Corel Paradox just before it changes the private directory. **setErrorCode** sets the error code to a nonzero value, which keeps this form open when the private directory changes. The code responds to the MenuChangedPriv menu command, issued by Corel Paradox just after it changes the private directory.

```
method menuAction(var eventInfo MenuEvent)


const
   kKeepFormOpen = UserMenu  ; UserMenu is an ObjectPAL constant.
endConst                     ; Any nonzero value keeps the form open.


   ; In a real app you'd declare and assign this variable elsewhere.
   okToChangePriv = True


   switch
      case eventInfo.id() = MenuFilePrivateDir :
         disableDefault                ; Block the default behavior.
         if okToChangePriv then
            setPrivDir("c:\\pdx\\mine") ; Set :PRIV: to hard-coded path.
         else
            return
         endIf


      case eventInfo.id() = MenuChangingPriv :
         eventInfo.setErrorCode(kKeepFormOpen)


      case eventInfo.id() = MenuChangedPriv :
         ; You may want to take some action after changing :PRIV:.
         ; This example just displays the new path.
         message(privDir())
         sleep(1000)


      otherwise : doDefault
   endSwitch
endMethod
```

setPrivDir example 2

The following example uses the **open** and the **menuAction** methods of a form to set the private directory before the form opens. In the form's built-in **open** method, **setPrivDir** changes the private directory to the same directory as the form. The ObjectPAL code in the **menuAction** prevents the form from closing during the change.

The following code is attached to the form's built-in **open** method:

```
;frm1 :: open
method open(var eventInfo Event)
   var
      f               Form
      dynPath   DynArray[] String
   endVar


   if eventInfo.isPreFilter() then
      ;// This code executes for each object on the form:
      f.attach()
      splitFullFileName(f.getFileName(), dynPath)
      setPrivDir(dynPath["Drive"] + dynPath["Path"])
   else
      ;// This code executes only for the form:
   endIf
endMethod
```

The following code is attached to the form's built-in **menuAction** method:

```
;frm1 :: menuAction
method menuAction(var eventInfo MenuEvent)
const
   kKeepFormOpen = UserMenu    ; UserMenu is an ObjectPAL constant.
endConst               ; Any nonzero value keeps the form open.


   if eventInfo.isPreFilter() then
      ;// This code executes for each object on the form:
      if eventInfo.id() = MenuChangingPriv then
         eventInfo.setErrorCode(kKeepFormOpen)
      endIf
   else
      ;// This code executes only for the form:
   endIf
endMethod
```

setWorkingDir procedure

Sets the working directory.

Syntax

```
setWorkingDir ( const path String ) Logical
```

Description

setWorkingDir sets the path of the current working directory. The following table gives examples of valid values for *path*:

Value of *path*	Example
Directory name	ORDERS
Full path	C:\\COREL\\SUITE8\\PARADOX\\APPS\\ORDERS\\
Relative path	..\\..\\ORDERS
Alias	:ORDERS:

By default, Corel Paradox closes all open windows before setting the working directory, and prompts you to save modified documents. Therefore, **setWorkingDir** does not take effect until all ObjectPAL code executes. You can keep a form open by adding code to its built-in **menuAction** method to trap for the MenuChangingWork menu command. If you do so, save any active documents before changing the working directory.

Use the following ObjectPAL MenuCommands constants to handle changes to the working folder:

Constant	Description
MenuFileWorkingDir	Issued when the user clicks Tools, Settings, Preferences, Database, Working Directory. Trap for this constant to prevent the user from changing the working directory.
MenuChangingWork	Issued before the working directory changes. Trap for this constant to keep a form open when changing the working directory.
MenuChangedWork	Issued after the working directory changes. Trap for this constant to determine whether the working directory has changed.

ObjectPAL also provides the constant MenuFilePrivateDir, issued when the user clicks Tools, Settings, Preferences, Database, Private Directory.

Examples

{button ,AL(`OPAL_TYPE_FILESYSTEM;OPAL_METH_FSSETPRIVDIR;OPAL_METH_FSWORKINGDIR;',0,"Defaultoverview",)} Related Topics

setWorkingDir procedure examples

Example1 Using a menu command to set the working directory

Example2 Using a form's **open** and **menuAction** methods

setWorkingDir example 1

The following example uses a menu command to change the working directory, and the resulting menu commands. When you click Tools, Settings, Preferences, Database, Working Directory, the code calls **disableDefault** to block the default behavior and prevent Corel Paradox from displaying the Set Working Directory dialog box. Next, this code tests the value of a Logical variable *okToChangeWork* (declared and assigned elsewhere). If *okToChangeWork* is True, it calls **setWorkingDir** to set the working directory (:WORK:) behind the scenes.

Also handles the MenuChangingWork menu command, issued by Corel Paradox just before it changes the working directory. The call to **setErrorCode** sets the error code to a nonzero value, which keeps the form open when the working directory changes. The code in this example responds to the MenuChangedWork menu command, issued by Corel Paradox just after it changes the working directory.

```
method menuAction(var eventInfo MenuEvent)
const
   kKeepFormOpen = UserMenu   ; UserMenu is an ObjectPAL constant.
endConst               ; Any nonzero value keeps the form open.


   ; In a real app you'd declare and assign this variable elsewhere.
   okToChangeWork = True


   switch
      case eventInfo.id() = MenuFileWorkingDir :
         disableDefault                ; Block the default behavior.
         if okToChangeWork then
            setWorkingDir("c:\\pdx\\mine") ; Set :WORK: to hard-coded path.
         else
            return
         endIf


      case eventInfo.id() = MenuChangingWork :
         eventInfo.setErrorCode(kKeepFormOpen)


      case eventInfo.id() = MenuChangedWork :
         ; You may want to take some action after changing :WORK:.
         ; This example just displays the new path.
         message(workingDir())
         sleep(1000)


      otherwise : doDefault
   endSwitch
endMethod
```

setWorkingDir example 2

The following example uses a form's **open** and **menuAction** methods to set the working directory before the form opens. In the form's built-in **open** method, **setWorkingDir** changes the current working directory to the same directory as the form. The ObjectPAL code in the **menuAction** prevents the form from closing during the change.

The following code is attached to the form's built-in **open** method:

```
;frm1 :: open
method open(var eventInfo Event)
var
   f                  Form
   dynPath   DynArray[]  String
endVar


if eventInfo.isPreFilter() then
   ;// This code executes for each object on the form:
else
   ;// This code executes only for the form:
   f.attach()
   splitFullFileName(f.getFileName(), dynPath)
   setWorkingDir(dynPath["Drive"] + dynPath["Path"])
endIf


endMethod
```

The following code is attached to the form's built-in **menuAction** method:

```
;frm1 :: menuAction
method menuAction(var eventInfo MenuEvent)
const
   kKeepFormOpen = UserMenu   ; UserMenu is an ObjectPAL constant.
endConst                      ; Any nonzero value keeps the form open.


   if eventInfo.isPreFilter() then
      ;// This code executes for each object on the form:
      if eventInfo.id() = MenuChangingWork then
         eventInfo.setErrorCode(kKeepFormOpen)
      endIf
   else
      ;// This code executes only for the form:
   endIf
endMethod
```

shortName method

Returns the short name of a file.

Syntax

```
shortName ( ) String
```

Description

After a successful **findFirst** or **findNext**, **shortName** returns the short name of the file whose name matches the pattern. A short name is the 8.3 filename stored in the file system.

Example

**{button ,AL(`OPAL_TYPE_FILESYSTEM;OPAL_METH_FSFDSP;OPAL_METH_FSTDSP;',0,"Defaultoverview",)
} Related Topics**

shortName example

The following example calls **findFirst** and **findNext** to locate tables in the current directory and then calls **shortName** to create a pop-up menu listing the filenames.

```
; showName::pushButton
method pushButton(var eventInfo Event)
var
   fs  FileSystem
   p  PopUpMenu
   tv  TableView
   choice, path  String
endVar


if fs.findFirst("*.db") then    ; if a *.db file exists
   p.addStaticText("Tables")     ; create a pop-up menu
   p.addSeparator()
   p.addText(fs.shortName())          ; use filenames in pop-up
   while fs.findNext()
      p.addText(fs.shortName())
   endWhile
   choice = p.show()             ; show the menu
   if not choice.isBlank() then  ; if user selected a table
      tv.open(choice)             ; display the selected table
   endif
endIf


endMethod
```

size method

Returns the size of a file.

Syntax

size ( ) LongInt

Description

size returns the size of a file, measured in bytes, after a successful **findFirst** or **findNext**.

Example

**{button ,AL(`OPAL_TYPE_FILESYSTEM;OPAL_METH_FSFDSP;OPAL_METH_FSTDSP;',0,"Defaultoverview",)
} Related Topics**

size example

The following example creates a dynamic array (DynArray) containing the filenames and sizes of the Corel Paradox tables in the current directory. The call to **view**, defined for the DynArray type, displays the information in a dialog box.

```
; demoButton::pushButton
method pushButton(var eventInfo Event)
var
  fs FileSystem
  da DynArray[] LongInt
endVar


if fs.findFirst("*.db") then
  da[fs.name()] = fs.size()
  while fs.findNext()
    da[fs.name()] = fs.size()
  endWhile
  da.view("Names and sizes")
else
  msgStop("*.db", "file not found.")
endIf


endMethod
```

splitFullFileName procedure

Breaks a full path name into its component parts.

Syntax

```
1. splitFullFileName ( const fullFileName String, var components DynArray[ ] String )
2. splitFullFileName ( const fullFileName String, var driveName String, var pathName String,
   var fileName String, var extensionName String )
```

Description

splitFullFileName divides a full path (obtained using **fullName**) into its component parts. **SplitFullFileName** does not return the values directly, but assigns them to variables that you declare and pass as arguments.

Syntax 1 assigns the returned values to a dynamic array that you must declare and pass as an argument. The DynArray has the following keys: DRIVE, PATH, NAME, and EXT.

Syntax 2 assigns the returned values to four String variables that you must declare and pass as arguments.

With both syntaxes, path components can include colons, periods, slashes, and backslashes. For example, if given C:\COREL\SUITE8\PARADOX\FORMS\ORDERS.FSL, **splitFullFileName** assigns values as follows:

DRIVE = C:

PATH = \COREL\SUITE8\PARADOX\FORMS\

NAME = ORDERS, and EXT = .FSL

The DRIVE variable (or key) stores everything up to and including the last colon in the filename. If the filename includes an alias, the alias is assigned to DRIVE. If the filename does not include a drive or an alias, an empty string is assigned to the DRIVE variable.

The PATH variable (or key) stores everything following the drive, up to and including the last backslash or slash. If the filename does not include a path, an empty string is assigned to the PATH variable. If a directory name in the path includes an extension, it is included in the PATH variable.

The NAME variable (or key) stores everything following the path, up to but not including the period that separates a filename from its extension. If the filename does not include a name, an empty string is assigned to the NAME variable.

The EXT variable (or key) stores everything following the filename, including the last period. If the filename does not include an extension, an empty string is assigned to the EXT variable.

Note

- The extension must be registered in the HKEY_CLASSES_ROOT section of the system registry to be return a value in this field. For more information about HKEY_CLASSES_ROOT, see your Windows documentation.

Examples

{button ,AL(`OPAL_TYPE_FILESYSTEM;OPAL_METH_FSFUNM;',0,"Defaultoverview",)} Related Topics

splitFullFileName procedure examples

Example1	Calling **fullName** first, then splitting the name
Example2	Splitting the name
Example3	Dealing with aliases

splitFullFileName example 1

The following example calls **fullName** to return the full name of the first form listed in the current directory. Then calls **splitFullFileName** to split the name into its component parts and store them in a dynamic array. The call to view displays the dynamic array.

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
  fs FileSystem
  splitName DynArray[] anytype
  fullFileName String
endVar


; if the customer.db file is in the sample directory
if fs.findFirst("c:\\Corel\\Suite8\\Paradox\\samples\\customer.db") then


  ; store the full filename to a variable
  fullFileName = fs.fullName()


  ; split filename into parts and store them in a DynArray
  splitFullFileName(fullFileName, splitName)


  ; display the component parts
  splitName.view("Split name")
endIf


endMethod
```

splitFullFileName example 2

The following example calls **splitFullFileName** to split the full name of a form into its component parts, then displays the path and the filename (without an extension) in dialog boxes.

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
   fs FileSystem
   driveName, pathName, fileName, extName String
endVar


   splitFullFileName("c:\\data\\sales\\stats.fsl", driveName, pathName, fileName, extName)
   pathName.view("Path name") ; displays the path
   fileName.view("Filename") ; displays the filename (no extension)


endMethod
```

splitFullFileName example 3

The following example displays a dialog box and prompts you to enter a filename. **splitFullFileName** splits the filename into its component parts and then displays the parts in dialog boxes.

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
   var
      stTestFileName,
      stPrompt,
      stDrive,
      stPath,
      stName,
      stExt         String
      dyFileName   DynArray[] String
   endVar


   stPrompt = "Enter a filename here."
   stTestFileName = stPrompt


   stTestFileName.view("Enter a filename to split:")


   if stTestFileName = stPrompt then
      ; User closed the dialog box without clicking OK,
      ; or clicked OK without typing a value.
      return
   else
      ; User typed a value and clicked OK.
      splitFullFileName(stTestFileName, dyFileName)
      dyFileName.view("DynArray")


      splitFullFileName(stTestFileName, stDrive, stPath, stName, stExt)
      stDrive.view("Drive")
      stPath.view("Path")
      stName.view("Name")
      stExt.view("Ext")
   endIf
endMethod
```

startUpDir procedure

Returns a string containing the path to your start-up directory (folder).

Syntax

```
startUpDir ( ) String
```

Description

startUpDir returns a string containing the path (including the drive letter) of the Corel Paradox start-up directory (folder).

Example

{button ,AL(`OPAL_TYPE_FILESYSTEM;OPAL_METH_FSPRIV;OPAL_METH_FSWORKINGDIR;',0,"Defaultoverview",)} Related Topics

startUpDir example

The following example opens a dialog box that displays the path to the Corel Paradox start-up directory (folder).

```
; thisButton::pushButton
method pushButton(var eventInfo Event)

msgInfo("Start-up directory", startUpDir())

endMethod
```

time method

Returns the time and date of a file's last modification.

Syntax

time ( ) DateTime

Description

time returns a DateTime value that represents the time and date of the file's last modification.

Example

**{button ,AL(`OPAL_TYPE_FILESYSTEM;OPAL_METH_FSFUNM;OPAL_METH_FSSIZE;',0,"Defaultoverview",)
} Related Topics**

time example

The following example calls **time** to return the time and date of the most recent modification to the *Customer* table. The code then compares the modification date with today's date and reports the results.

```
method pushButton(var eventInfo Event)
   var
      fs FileSystem
   endVar

   if fs.findFirst("customer.db") then
      if fs.time() < DateTime(today()) then
         message("old version")
      else
         message("new version")
      endif
   endIf
endMethod
```

totalDiskSpace method

Returns the total capacity of a specified drive, measured in bytes.

Syntax

```
totalDiskSpace ( const driveLetter String ) LongInt
```

Description

totalDiskSpace returns the total number of bytes the specified drive can hold. You can specify a drive using a letter (C) or a letter and a colon (C:).

Example

{button ,AL(`OPAL_TYPE_FILESYSTEM;OPAL_METH_FSFDSP;',0,"Defaultoverview",)} **Related Topics**

totalDiskSpace example

The following example calls **totalDiskSpace** and **freeDiskSpace** to calculate the amount of space available. The code stores the information in a dynamic array and then calls the appropriate **view** method to display the information in a dialog box.

```
; spaceUsed::pushButton
method pushButton(var eventInfo Event)
var
  fs FileSystem
  da DynArray[] LongInt
endVar


da["Total space"] = fs.totalDiskSpace("C")
da["Free space"] = fs.freeDiskSpace("C")
da["Space in use"] = da["Total space"] - da["Free space"]
da.view("Drive C")


endMethod
```

windowsDir procedure

Returns the path to the WINDOWS directory (folder).

Syntax

```
windowsDir ( ) String
```

Description

windowsDir returns the path to the WINDOWS directory.

Example

{button ,AL(`OPAL_TYPE_FILESYSTEM;OPAL_METH_FSWSDI;OPAL_METH_FSWORKINGDIR;OPAL_METH_FSPRIV;OPAL_METH_FSSTART;',0,"Defaultoverview",)} Related Topics

windowsDir example

The following example reads WIN.INI from drive B and copies it to the WINDOWS folder on the default drive.

```
; copyWinIni::pushButton
method pushButton(var eventInfo Event)
var
   fs  FileSystem
   fileName, destName  String
endVar


fileName = "\\win.ini"


fs.setDrive("B")
if fs.findFirst(fileName) then
   destName = windowsDir() + fileName
   fs.copy(fileName, destName)
endIf


endMethod
```

windowsSystemDir procedure

Returns the path to the Windows System directory (folder).

Syntax

windowsSystemDir ( ) String

Description

windowsSystemDir returns the path to the Windows System directory.

Example

{button ,AL(`OPAL_TYPE_FILESYSTEM;OPAL_METH_FSWDIR;',0,"Defaultoverview",)} **Related Topics**

windowsSystemDir example

The following example reads SPECIAL.DRV from drive B and copies it to the Windows System directory (folder) on the default drive.

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
   fs  FileSystem
   fileName, destName  String
endVar


fileName = "\\special.drv"


fs.setDrive("B")
if fs.findFirst(fileName) then
   destName = windowsSystemDir() + fileName
   fs.copy(fileName, destName)
endIf


endMethod
```

workingDir procedure

Returns the name of the working directory.

Syntax

workingDir ( ) String

Description

workingDir returns the name (including the path) of the working directory.

Example

{button ,AL(`OPAL_TYPE_FILESYSTEM;OPAL_METH_FSPRIV;OPAL_METH_FSWDIR;OPAL_METH_FSWSDI;',0,"Defaultoverview",)} **Related Topics**

workingDir example

The following example displays a message that contains the path to the working directory:

```
; thisButton::pushButton
method pushButton(var eventInfo Event)

message("Working directory is: " + workingDir())

endMethod
```

Disk errors

When a method fails because of a disk error, the error code constant is peDiskError and the error message is, "A disk error occurred" plus one of the following strings:

"Invalid function number."

"The file could not be found."

"The directory path could not be found."

"No file handle available."

"Access to this file is denied. It is read only or a directory."

"Invalid handle."

"Memory control blocks have been damaged."

"Insufficient memory to allocate file structures."

"Invalid memory block address."

"Invalid environment."

"Invalid format."

"Invalid file access byte."

"Invalid data."

"Invalid drive."

"Cannot remove the current directory."

"Not the same device."

"No more files match the wildcard specification."

"Cannot write to a write-protected disk."

"Unknown unit."

"The drive is not ready."

"Command is not recognized."

"Checksum error (Bad CRC)."

"Invalid request structure length."

"File seek error."

"Unknown media type."

"Sector not found."

"Out of paper."

"An error occurred while trying to write to the disk."

"An error occurred while trying to read from the disk."

"General DOS error."

"File sharing violation."

"File lock violation."

"Invalid disk change."

"File control blocks unavailable."

"Sharing buffer overflow."

"Bad code page."

"Handle EOF."

"The disk is full."

"Device is not supported."

"Device is not listening."

"Duplicate name."

"Invalid network path."

"The network is busy."

"The device does not exist."

"Too many commands."

"Adapter error."
"Invalid network response."
"Network error."
"Adapter is incompatible."
"The print queue is full."
"Out of spool space."
"Print job was canceled."
"The network name was deleted."
"Your access to the network is denied."
"Invalid device type."
"Invalid network name."
"Too many names."
"Too many sessions."
"Sharing pause."
"Request not accepted."
"Redirection pause."
"The file already exists."
"Duplicate file control blocks."
"Cannot create the specified directory."
"DOS critical error."
"Out of structures. Cannot perform operation."
"Drive is already assigned."
"Invalid password."
"Invalid parameter."
"Network write error."
"Comp command is not loaded."
"The mode specification is invalid."
"Cannot write to the file because it was opened in read-only mode."

{button ,AL(`OPAL_TYPE_FILESYSTEM;OPAL_TYPE_FILESYSTEM;;',0,"Defaultoverview",)} **Related Topics**

Form type

A Form variable provides a handle for working with a Corel Paradox form. Form type methods let you

- load a form in a Form Design window and save a design
- open and close a form
- attach to an open form
- work with tables in a data model
- work with table aliases
- enumerate object names, properties, and source code for methods
- determine and change the position of a form, as well as maximize or minimize the form
- send events to a form, such as a **mouseUp** or **keyPhysical**
- get and set methods for a form

The Form type is the base type from which the other display manager types (for example, Report) are derived. Many of the methods listed in this section are also used by the Application, Report, and TableView types.

Methods in the Form type

action
attach
bringToTop
close
create
delayScreenUpdates
deliver
design
disableBreakMessage
disablePreviousError
dmAddTable
dmAttach
dmBuildQueryString
dmEnumLinkFields
dmGet
dmGetProperty
dmHasTable
dmLinkToFields
dmLinkToIndex
dmPut
dmRemoveTable
dmResync
dmSetProperty
dmUnlink
enumDataModel
enumSource
enumSourceToFile
enumTableLinks
enumUIObjectNames
enumUIObjectProperties
formCaller
formReturn
getFileName

getPosition
getProtoProperty
getSelectedObjects
getStyleSheet
getTitle
hide
hideToolbar
isCompileWithDebug
isDesign
isMaximized
isMinimized
isToolbarShowing
isVisible
keyChar
keyPhysical
load
maximize
menuAction
methodDelete
methodEdit
methodGet
methodSet
minimize
mouseDouble
mouseDown
mouseEnter
mouseExit
mouseMove
mouseRightDouble
mouseRightDown
mouseRightUp
mouseUp
moveToPage
open
openAsDialog
postAction
run
save
saveStyleSheet
selectCurrentTool
setCompileWithDebug
setIcon
setMenu
setPosition
setProtoProperty
setSelectedObjects
setStyleSheet
setTitle
show

showToolbar
wait
windowClientHandle
windowHandle
writeText

Print related ObjectPAL methods and examples

action method/procedure

Performs an action command.

Syntax

```
action ( const actionId SmallInt ) Logical
```

Description

action performs the function represented by the constant *actionId*, where *actionId* is a constant in one of the following action classes:

- ActionDataCommands
- ActionEditCommands
- ActionFieldCommands
- ActionMoveCommands
- ActionSelectCommands

You can also use **action** to send a user-defined action constant to a built-in **action** method. User-defined action constants are simply integers that don't interfere with any of ObjectPAL's constants. You can use user-defined action constants to signal other parts of an application. For example, assume that the Const window for a form declares a constant named *myAction*. In the built-in **action** method for a page on the form, you might check the value of every incoming ActionEvent (with the **id** method); if the value is equal to *myAction*, you can respond to that action accordingly. Corel Paradox's default response for user-defined action constants is simply to pass the action to the **action** method.

This **action** method is distinct from the built-in **action** method for a form or for any other UIObject. The built-in **action** method for an object responds to an action event; this method causes an ActionEvent.

Note

- When you call the **action** method as a procedure, the form dispatches it to the object represented by *self*. The event bubbles through the containership hierarchy until the event either reaches an object that can handle the action or the event reaches the form. If the event reaches the form, and the action is a data action, the form sends the event to the master table for the form.

Example

{button ,AL(`OPAL_TYPE_FORM;OPAL_METH_UIACTI;',0,"Defaultoverview",)} Related Topics

action example

In the following example, a form named *Sitenote* contains field objects bound to the *Sites* table. The current form contains a button named *openEditSites*; the **pushButton** method for *openEditSites* opens *Sitenote*, starts Edit mode, and waits for *Sitenote* to be closed:

```
; openEditSites::pushButton
method pushButton(var eventInfo Event)
var
   siteForm  Form
endVar
siteForm.open("Sitenote.fsl")  ; open Sitenote
siteForm.action(DataBeginEdit) ; start Edit mode on siteForm
message("To return, close Sitenote form.")
siteForm.wait()                ; this form will be inactive until
                               ; Sitenote returns
siteForm.close()               ; this form must close Sitenote
endMethod
```

attach method

Associates a Form variable with an open form.

Syntax

```
attach ( [ const formTitle String ] ) Logical
```

Description

attach associates a Form variable with an open form. You can use *formTitle* to specify a form's title, or you can omit *formTitle* to attach to the form where **attach** is executing. This method returns True if successful; otherwise, it returns False.

Note

- The argument *formTitle* specifies a form's title as displayed in the Title Bar (for example, Orders), not the form's filename or UIObject name. You can specify a form's title interactively by right-clicking the form's Title Bar, choosing Window Style, and entering a value in the Window Style dialog box. You can specify a title in ObjectPAL by setting a form's Title property, or by calling **setTitle**.

Example

{button ,AL(`OPAL_TYPE_FORM;OPAL_METH_FOGETT;OPAL_METH_FOSTIT;OPAL_METH_FOOPEN;',0,"Defaultoverview",)} Related Topics

attach example

In the following example, a form has two buttons: *openSites* and *attachToSites*. The **pushButton** method for *openSites* opens the *Sitenote* form. The **pushButton** method for *attachToSites* attaches the form variable *sitesForm* to the open form by way of the form's current title. In this case, the form title wasn't changed; therefore *attachToSites* can attach to *Sitenote* using the default title. Once attached, the **pushButton** method uses the *sitesForm* handle to minimize, maximize, and restore *Sitenote*.

The following code is attached to the **pushButton** method for *openSites*:

```
; openSites::pushButton
method pushButton(var eventInfo Event)
var
   sitesForm Form
endVar
sitesForm.open("Sitenote")
sitesForm.Title = "Notes" ; Set the form's title.


endMethod
```

The following code is attached to the **pushButton** method for *attachToSites*:

```
; attachToSites::pushButton
method pushButton(var eventInfo Event)
var
   sitesForm  Form
endVar


; Attach to Sitenote by its title (Notes).
; Note that this won't work:   sitesForm.attach("Sitenote")
if not sitesForm.attach("Notes") then
      errorShow()
      return
endIf


; cycle through sizes
sitesForm.minimize()               ; minimize the form
sleep(2000)                        ; pause
sitesForm.maximize()               ; maximize the form
sleep(2000)                        ; pause
sitesForm.show()                   ; restore to original size
endMethod
```

bringToTop method/procedure

Brings the window to the top of the display stack and makes it active.

Syntax

```
bringToTop ( )
```

Description

When several windows are displayed they seem to overlap and give the appearance of layers. Use **bringToTop** to display a window on the top of the stack and not overlapped by any other windows. **bringToTop** makes a form the active window.

If a hide statement has made a form invisible, **bringToTop** makes it visible again.

Example

{button ,AL(`OPAL_TYPE_FORM;OPAL_METH_FOIVIS;OPAL_METH_FOHIDE;OPAL_METH_FOSHOW;OPAL_TYPE_APPLICATION;',0,"Defaultoverview",)} **Related Topics**

bringToTop example

In the following example, the **pushButton** method for a button named *openSeveral* opens the *Sitenote* form and then opens a Table window for the *Orders* table. The Table window, *orderTV*, opens over the *Sitenote* form, *siteForm*. The method pauses for a few seconds and then makes *siteForm* the topmost layer:

```
; openSeveral::pushButton
method pushButton(var eventInfo Event)
var
   siteForm  Form
   orderTV   TableView
endVar
siteForm.open("Sitenote.fsl")   ; opens Sitenote form
orderTV.open("orders")          ; opens Orders over Sitenote
message("About to make the Sitenote form the highest layer.")
beep()
sleep(5000)                   ; pause
siteForm.bringToTop()         ; make Sitenote highest layer


endMethod
```

close method/procedure

Closes a window.

Syntax

```
1. ( Method ) close ( )
2. ( Procedure ) close ( [ const returnValue AnyType ] )
```

Description

close closes a window as if the user has chosen Close from the Control menu.

Example

{button ,AL(`OPAL_TYPE_FORM;OPAL_METH_FOOPEN;',0,"Defaultoverview",)} **Related Topics**

close example

The following example uses **close** to return a value to a form that called it with **wait**. Assume a form contains a button called *btn1*. A second form contains two buttons called *btnReturnOK* and *btnReturnCancel*. The first form opens the second form and waits for one of three values: OK, Cancel, or False. OK and Cancel are returned from the two buttons on the second form (see the following code) and False is returned if the user closes the second form without pressing a button. The first form processes the user's selection in a **switch** statement that calls one of three custom methods (assumed to be defined elsewhere).

The following code is attached to the button *btn1* in the first (calling) form.

```
;frm1.btn1 :: pushButton
method pushButton(var eventInfo Event)
   var
      f   Form        ;Declare form variable.
      s   String      ;Declare string value.
   endVar


   f.open("wait2")      ;Open form that will return string.
   s = string(f.wait())      ;Wait for value from other form.
   s.view("Returned value")   ;View returned value.


   ;Process returned value using custom methods defined elsewhere.
   switch
      case s = "OK"    : cmOK()       ;User pressed the OK button.
      case s = "Cancel": cmCancel();User pressed the Cancel button.
      case s = "False" : cmNone()  ;User closed form, no button pressed.
   endSwitch
endmethod
```

The following code is attached to the button *btnReturnOK* in the second (called) form:

```
;frm2.btnReturnOK :: pushButton
method pushButton(var eventInfo Event)
   close("OK")         ;Close & return OK.
endmethod
```

The following code is attached to the button *btnReturnCancel* in the second (called) form:

```
;frm2.btnReturnCancel :: pushButton
method pushButton(var eventInfo Event)
   close("Cancel")      ;Close & return Cancel.
endmethod
```

create method

Creates a blank form in a Form Design window.

Syntax

create ( ) Logical

Description

create opens a blank form and leaves it in a Form Design window. You can use the UIObject type methods **create** and **methodSet** to place objects in the new form and attach methods to them. You can attach methods to the form using the Form type method **methodSet**. Use the Form type method **run** to open the form in a Form window.

Example

{button ,AL(`OPAL_TYPE_FORM;OPAL_METH_FODESIGN;OPAL_METH_FOLOAD;OPAL_METH_FOOPEN;OPAL_METH_FOMETHODGET;OPAL_METH_FOMETHODSET;OPAL_METH_FORUN;OPAL_METH_UICREATE;OPAL_METH_UIMETHODGET;OPAL_METH_UIMETHODSET;',0,"Defaultoverview",)} Related Topics

create example

In the following example, the **pushButton** method for a button named *createAForm* creates a new form with the **create** method and sets the value of the new form's **mouseUp** method with **setMethod**. The **pushButton** method for *createAForm* then saves the new form to a file named NEWHELLO.FSL, runs the form, and calls the new form's **mouseUp** method (supplying the correct arguments). The **mouseUp** method for the *Newhello* form opens a dialog box that displays Hello. After the dialog box is closed (by the user), the **pushButton** method for *createAForm* closes the *Newhello* form.

```
; createAForm::pushButton
method pushButton(var eventInfo Event)
var
   newForm Form
endVar
newForm.create()              ; create a new blank form (a Form Design window)
newForm.methodSet("mouseUp",  ; set the mouseUp method for the form
"method mouseUp(var eventInfo MouseEvent)
msgInfo(\"Greetings\", \"Hello\")
endMethod")                   ; backslashes delimit embedded quotes
newForm.save("newhello")      ; save the form
newForm.run()                 ; run the new form (View Data window)
                              ; call the mouseUp method for the form
newForm.mouseUp(100, 100, LeftButton )  ; dialog box displays "Hello"
newForm.close()               ; close the form
endMethod
```

delayScreenUpdates procedure

Turns delayed screen updates on or off.

Syntax

```
delayScreenUpdates ( const yesNo Logical )
```

Description

delayScreenUpdates postpones or enables the redrawing of areas of the screen. You must specify Yes or No in *yesNo*. Specifying Yes delays screen updates (redraws) until the system yields or is idle. This can increase performance in operations that frequently refresh the display (e.g., when using ObjectPAL to add items to a list). Specifying No allows screen updates to occur without delay.

For some operations, you won't notice a difference when **delayScreenUpdates** is set to Yes.

Example

{button ,AL(`OPAL_TYPE_FORM;OPAL_METH_SYSLEEP;',0,"Defaultoverview",)} Related Topics

delayScreenUpdates example

The following two methods override the **pushButton** methods for their respective buttons. The *drawOneByOne* button draws a number of boxes without changing **delayScreenUpdates**. The *drawAllAtOnce* button draws the same number of boxes, to a different location, but first sets **delayScreenUpdates** to Yes. When this code runs, you'll see the boxes created by *drawOneByOne* appear one at a time, but still rapidly. The boxes created by *drawAllAtOnce* are created behind the scenes, which causes a short pause then they all appear at the same time.

```
; drawOneByOne::pushButton
method pushButton(var eventInfo Event)
var
  ui UIObject
endVar


; delayScreenUpdates(No) is the default
; Create and display a set of boxes, showing them as
; they're created.
for i from 750 to 2550 step 300
  for j from 750 to 2550 step 300
    ui.create(boxTool, i, j, 150, 150)
    ui.Color = Blue
    ui.Visible = Yes
  endfor
endfor
endMethod
```

The *drawAllAtOnce* button on the same form creates the same number of boxes, but does so with **delayScreenUpdates** set to Yes. On very fast machines, you still may not be able to see the difference.

```
; drawAllAtOnce::pushButton
method pushButton(var eventInfo Event)
var
  ui UIObject
endVar


delayScreenUpdates(Yes)
; This code will create all boxes and then display
; them all at once.
for i from 4950 to 6750 step 300
  for j from 750 to 2550 step 300
    ui.create(boxTool, i, j, 150, 150)
    ui.Color = Red
    ui.Visible = Yes
  endfor
endfor
; reset to default
delayScreenUpdates(No)


endMethod
```

deliver method

Delivers a form.

Syntax

deliver ( ) Logical

Description

deliver behaves like Format, Deliver. This method saves a copy of a form with an .FDL extension, which prevents users from editing the form in the Form Design window. Users can open the form only in a Form window. Switching to the Form Design window on an open, delivered form is also prohibited.

Corel Paradox opens saved forms before delivered forms with the same name. For example, suppose the working directory contains ORDERS.FSL (a saved form) and ORDERS.FDL (a delivered form).

The following statement opens the saved form, ORDERS.FSL.

```
ordersForm.open("ORDERS") ; Opens :WORK:ORDERS.FSL.
```

To specify a delivered form, include the .FDL extension. For example,

```
ordersForm.open("ORDERS.FDL") ; Opens the delivered form.
```

Example

{button ,AL(`OPAL_TYPE_FORM;OPAL_METH_FOSAVE;',0,"Defaultoverview",)} Related Topics

deliver example

In the following example, the *createDeliver* button creates a new form, saves it to the name *Newhello* and then delivers it (which saves a version as NEWHELLO.FDL). When the method attempts to load the form in a Form Design window, load returns False because a delivered form can't be loaded in a Form Design window.

```
; createDeliver::pushButton
method pushButton(var eventInfo Event)
var
   newForm Form
endVar
newForm.create()              ; create a new blank form (a Form Design window)
newForm.save("newhello")      ; save the form
newForm.deliver()             ; deliver the newly created form
newForm.close()               ; close the form
if NOT newForm.load("newhello.fdl") then  ; load will return False
   errorShow("Can't load a delivered form.")
endif
endMethod
```

design method

Switches a form from the Form window to the Form Design window.

Syntax

```
design ( ) Logical
```

Description

design switches a form from the Form window to the Form Design window. This method works only with saved forms (.FSL); it does not work with delivered forms (.FDL).

Use **run** to switch from the Form Design window to the Form window.

Note

- Some form actions are especially processor-intensive. In some situations, you might need to follow a call to **open**, **load**, **design**, or **run** with a call to **sleep**. For more information, see the **sleep** method in the System type.

Example

{button ,AL(`OPAL_TYPE_FORM;OPAL_METH_FOCREATE;OPAL_METH_FOLOAD;OPAL_METH_FOOPEN;OPAL_METH_FORUN;',0,"Defaultoverview",)} Related Topics

design example

The following example uses a custom procedure to force a form (specified by its title) into design mode.

```
proc forceDesign(const foTemp Form) Logical
if foTemp.isDesign() then
   return True
else
   return foTemp.design()
endIf
endProc
```

disableBreakMessage procedure

Prevents program interruption by CTRL + Break.

Syntax

```
disableBreakMessage ( const yesNo Logical ) Logical
```

Description

disableBreakMessage lets you prevent or allow the user to interrupt a running program with CTRL + Break.

Example

{button ,AL(`OPAL_TYPE_FORM;OPAL_METH_FODISABLEPREVIOUSERROR;OPAL_METH_SYDEBUG;',0,"Defaultoverview",)} **Related Topics**

disableBreakMessage example

In the following example, assume a form contains a table frame bound to the *Orders* table.

The following code prevents the loop from being interrupted by a CTRL + Break.

```
; throughTable::pushButton
method pushButton(var eventInfo Event)
; just a loop to test CTRL-breaking out of
disableBreakMessage(Yes)     ; don't allow a CTRL + Break
while NOT ORDERS.atLast()
  ORDERS.action(DataNextRecord)
endwhile
endMethod
```

disablePreviousError procedure

Specifies whether you have access to the Previous Error dialog box.

Syntax

```
disablePreviousError ( const yesNo Logical ) Logical
```

Description

By default, when you move the pointer over the Status Bar, the pointer changes shape; you can then click the Status Bar to display the Previous Error dialog box (if error information is available). If *yesNo* is Yes (or True), **disablePreviousError** prevents this behavior; otherwise it restores the default behavior.

Returns True if successful; otherwise, returns False. This setting remains in effect (and affects all forms) as long as Corel Paradox is running. The default behavior is restored the next time you start Corel Paradox.

Example

{button ,AL(`OPAL_TYPE_FORM;OPAL_METH_SYDEBUG;OPAL_METH_FODISABLEBREAKMESSAGE;',0,"Defaultoverview",)} Related Topics

disablePreviousError example

The following example uses **disablePreviousError** in a script named *InitApp* to prevent user access to the Previous Error dialog box:

```
; InitApp::run
method run(var eventInfo Event)
   disablePreviousError(Yes)
   openMainForm() ; Call a custom method to open the main application form.
endMethod
```

dmAddTable method/procedure

Adds a table to a form's data model.

Syntax

```
dmAddTable ( const tableName String ) Logical
```

Description

dmAddTable adds the table *tableName* to a form's data model, where *tableName* is a valid table name. This method returns True if successful; otherwise, it returns False.

Example

{button ,AL(`OPAL_TYPE_FORM;OPAL_METH_FODMHASTABLE;OPAL_METH_FODMREMOVETABLE;',0,"Defaultoverview",)} Related Topics

dmAddTable example

In the following example, a form contains a button named *toggleSites* and a list field named *showSiteNames*. The list data for the *showSiteNames* field is set with the DataSource property of its list object, *ListNames*. The **pushButton** method for *toggleSites* checks to see if the *Sites* table is in the data model for the form. If so, the reference to *Sites* is removed from the DataSource property of *ListNames* and then *Sites* is removed from the data model; otherwise, the *Sites* table is added to the data model and the DataSource property of *ListNames* is set to the *Site Name* field of *Sites*.

The following code is attached to the **pushButton** method of *toggleSites*:

```
; toggleSites::pushButton
method pushButton(var eventInfo Event)
      ; toggle Sites.db in and out of the data model
if dmHasTable("Sites") then    ; is Sites in data model?
      ; if so, remove dependencies and then remove table
      ; remove Sites as source from showSiteNames.ListNames
   showSiteNames.ListNames.DataSource = ""
   showSiteNames.Visible = False
      ; remove Sites from the data model
   dmRemoveTable("Sites")
   whichTable = ""
else
      ; if not already in data model and then add Sites
   dmAddTable("Sites")
      ; set the data for the list from the Sites table
   showSiteNames.ListNames.DataSource = "[Sites.Site Name]"
   showSiteNames.Visible = True
   whichTable = "Sites"
endIf

endMethod
```

dmAttach method/procedure

Associates a TCursor variable with a table in the form's data model.

Syntax

```
dmAttach ( tc TCursor, const tableName String ) Logical
```

Description

dmAttach associates the TCursor variable *tc* with the table *tableName* in the form's data model, where *tableName* is either a valid table name or a table alias. This method returns True if successful; otherwise it returns False.

Example

{button ,AL(`OPAL_TYPE_FORM;OPAL_METH_FODMHASTABLE;OPAL_METH_TCDMATTACH;OPAL_METH_TCOPEN;IDH_MULT_TABALIAS@PDOX.HLP;',0,"Defaultoverview",)} Related Topics

dmAttach example

The following example demonstrates how to use **dmAttach** and **dmResync** to keep two forms synchronized. Both forms have the *Customer* table in their data models. When the user moves from the first form *frm1* to the second form *frm2*, a form variable *f* is used to attach back to the first form and **dmAttach** is used to attach to the appropriate table in its data model. **dmResync** is used to move to the same record as the first form.

```
;Frm2.pge1 :: setFocus
method setFocus(var eventInfo Event)
var
   f   Form         ;Declare a form variable.
   tc   TCursor   ;Declare a TCursor variable.
endVar


if f.attach("dmAttach2") then   ;Attach to other form.
   f.dmAttach(tc, "Customer.db")   ;Attach tc to a table in the
   dmResync("Customer.db", tc)   ;data model of the other form.
                              ;Then sync the two forms.
endIf
endMethod
```

dmBuildQueryString method/procedure

Builds a query string based on the data model of a form.

Syntax

```
dmBuildQueryString ( var queryString String ) Logical
```

Description

dmBuildQueryString creates a query string *queryString* based on the data model of a form. The query built by **dmBuildQueryString** creates checked example elements for all the link fields in the data model. The form's data model must have a linked table. **dmBuildQueryString** returns True if it is successful; otherwise, it returns False.

Examples

{button ,AL(`OPAL_TYPE_FORM;OPAL_METH_FODMHASTABLE;OPAL_METH_FOENUMDATAMODEL;OPAL_METH_QUREADFROMSTRING;',0,"Defaultoverview",)} **Related Topics**

dmBuildQueryString examples

Example1	Displaying a query string based on an existing data model
Example2	A **pushButton** method uses **dmBuildQueryString** to generate a query string

dmBuildQueryString example 1

The following example assumes a data model has the *Customer* and *Orders* tables linked on the CustomerNo field. The code displays a query string based on that data model.

```
method pushButton(var eventInfo Event)
   var
      stQBE String
   endVar
   dmBuildQueryString(stQBE)
   stQBE.view("Query String")


{
Displays the following string:
Query

::C:\COREL\SUITE8\PARADOX\BLDNOTES\CUSTOMER.DB|CustomerNo   |
                                  |Check _join1!|

::C:\COREL\SUITE8\PARADOX\BLDNOTES\ORDERS.DB|CustomerNo  |
                                |Check _join1|

EndQuery
}
endMethod
```

dmBuildQueryString example 2

The following example assumes a form contains a button named *btnDMQuery*. The **pushButton** method for *btnDMQuery* uses **dmBuildQueryString** as a procedure to generate a query string in *s*. **readFromString** is called to assign the string to a Query variable and the method runs the query and opens a Table window for the *Answer* table.

```
;btnDMQuery :: pushButton
method pushButton(var eventInfo Event)
   var
      s   String
      tv   TableView
      qVar   Query
   endVar


   dmBuildQueryString(s)
   qVar.readFromString(s)
   if qVar.executeQBE() then
      tv.open(":PRIV:ANSWER.DB")
   else
      errorShow()
      return
   endIf
endMethod
```

dmEnumLinkFields method/procedure

Lists the fields that link two tables.

Syntax

```
dmEnumLinkFields ( var masterTable String, var masterFields Array[ ] String, const detailTable String, var detailFields Array[ ] String, var detailIndex String ) Logical
```

Description

dmEnumLinkFields lists the fields that link the tables named in *masterTable* and *detailTable*. You must supply a table name or table alias for *detailTable*. This method assigns values to the other variables (passed as arguments) as follows:

Variable	Assigned value
masterTable	The name of the master table. Blank if the table specified in *detailTable* has no master table.
masterFields	Names of the linking fields in the master table. Blank if the table specified in *detailTable* has no master table.
detailFields	Names of the linking fields in the detail table. Blank if the table specified in *detailTable* has no master table. If the detail table is a dBASE table and uses an expression index, the expression is returned in angled brackets. Examples: <FIRSTNAME + LASTNAME> means an expression index based on the fields named FIRSTNAME and LASTNAME; <FIRSTNAME + LASTNAME;QTY > 1> means an expression index based on the fields named FIRSTNAME and LASTNAME with QTY > 1 as a subset condition.
indexName	Name of the index used by the detail table. Blank if the table specified in *detailTable* is not using an index. If the detail table is a dBASE table, you can use **dmGetProperty** to get the associated tag name, if any.

The tables must already be in the specified data model. This method returns True if successful; otherwise, it returns False.

Example

{button ,AL(`OPAL_TYPE_FORM;OPAL_METH_FODMHASTABLE;IDH_MULT_TABALIAS@PDOX.HLP;',0,"Defaultoverview",)} Related Topics

dmEnumLinkFields example

In the following example, assume that a form's data model links the *Customer* and *Orders* tables on the CustomerNo field, with the *Orders* table as the detail table. The tables do not use secondary indexes.

```
method pushButton(Var eventInfo Event)
   var
      mAr, dAr Array[] String
      m, d, inx String
   endVar


   d = "orders"
   dmEnumLinkFields(m, mAr, d, dAr, inx)
   m.view("Master table name")    ; Displays CUSTOMER.DB
   mAr.view("Master link fields") ; Displays Customer No
   d.view("Detail table name")    ; Displays ORDERS.DB
   dAr.view("Detail link fields") ; Displays Customer No
   inx.view("Index name")         ; Displays Customer No
endMethod
```

dmGet method/procedure

Retrieves a field value from a table in the data model.

Syntax

```
dmGet ( const tableName String, const fieldName String, var datum AnyType ) Logical
```

Description

dmGet provides access to table data in the form's data model. **dmGet** writes to *datum*, a field value from a specified table. The table specified by *tableName* must be the name or table alias of a table in the form's data model. *fieldName* must be a field in *tableName*.

Example

{button ,AL(`OPAL_TYPE_FORM;OPAL_METH_FODMPUT;',0,"Defaultoverview",)} Related Topics

dmGet example

In the following example, a form contains a table frame bound to the *Sites* table. The table frame contains only two fields: Site No and Site Name. The **pushButton** method for a button named *getHighlight* uses **dmGet** to find the value of the Site Highlight field for the active record. The method then displays the Site Highlight value in a dialog box and asks the user whether to change the value. If the user answers Yes in the dialog box, the method shows the original value for Site Highlight in a dialog box and prompts the user for a new value. The method then uses **dmPut** to write the changed value back to the *Sites* table.

```
; getHighlight::pushButton
method pushButton(var eventInfo Event)
var
  siteHighlight AnyType
  qAnswer       String
endVar
; get the value in the Site Highlight field for the active record
if dmGet("Sites", "Site Highlight", siteHighlight) then
 ; show the highlight and ask the user whether to change it
 qAnswer = msgQuestion("Change Highlight?",
              "At site " + SITES.Site_Name +
              " the highlight is " +
              String(siteHighlight) + ". Change highlight?")
 if qAnswer = "Yes" then
   ; check for Edit mode
   if thisForm.Editing <> True then
     action(DataBeginEdit)
   endif
   ; ask user to replace existing highlight value in View dialog box
   siteHighlight.view("Enter a new highlight:")
   ; write the changed highlight back to the Site Highlight field
   dmPut("Sites", "Site Highlight", siteHighlight)
 endif
else
  msgStop("Sorry", "Couldn't find the highlight for this site.")
endif
endMethod
```

For information on table aliases, see Table Aliases in the Corel Paradox User's Guide help.

dmGetProperty method/procedure

Returns the value of a specified table property.

Syntax

```
1. dmGetProperty ( const tableName String, const propertyName String ) AnyType

2. dmGetProperty ( const tableName String, const propertyName String, var value AnyType )
Logical
```

Description

Returns the value of a property *propertyName* of the table *tableName* in the specified data model. The value of *tableName* must be a valid table name or a table alias.

The return value depends on the value of *propertyName* that you supply from the following:

This value	Returns
AutoAppend	True if Auto Append is set to True for the table; otherwise, it returns False.
Editing	True when a form is in Edit mode, or a field object is active and being edited; otherwise, it returns False.
Flyaway	True when a record has moved to its sorted position in a table; otherwise, it returns False.
FullName	The full filename (as a string, including path or alias) of the table.
Index	The name of the index (as a string) that is currently used to view the table. For a child table, it returns the name of the index chosen in the link diagram. For a master table or unlinked table, it returns the setting of ORDER/RANGE. It returns an empty string when the primary key is used.
Inserting	True when a record is being inserted anywhere in a form; otherwise, it returns False.
LinkType	A string describing the way the table relates to its master table: None, One-to-one, or One-to-many.
Locked	True when the table bound to a design object is locked; otherwise, it returns False.
Name	The table's alias (as a string) if it exists; otherwise, returns an empty string.
Next	The name (as a string) of the next object in the same container.
One-to-many	The name (as a string) of the first detail table linked 1:M to this table.
One-to-one	The name (as a string) of the first detail table linked 1:1 to this table.
Parent	The table name (as a string) of this table's master in the data model.
Read-only	True if READONLY is set to True for the table; otherwise, it returns False.
Refresh	True when data displayed onscreen is being changed, either across a network (by an ObjectPAL statement) or by a user action; otherwise, it returns False.
StrictTranslation	True if STRICT TRANSLATION is set to True for the table; otherwise, it returns False.
TagName	The tag name (as a string) for the current dBASE index (if any); otherwise, it returns an empty string.
Touched	True when the user has made changes to data not yet committed.

Syntax 1 returns the property value directly.

Syntax 2 assigns the value to *value*, an AnyType variable that you declare and pass as an argument. Syntax 2 returns True if the method succeeds; otherwise, it returns False.

For both syntaxes, **dmGetProperty** returns False if *tableName* is not in the data model, or if the value of *propertyName* is not one of the strings listed earlier.

The value of *tableName* must be a valid table name or a table alias .

If *propertyValue* = Name this method returns the table's alias (as a string) if it exists; otherwise, it returns an empty string.

If *propertyValue* = FullName this method returns the full filename (including path or alias) of the table.

Example

{button ,AL(`OPAL_TYPE_FORM;OPAL_METH_FODMSETPROPERTY;',0,"Defaultoverview",)} Related Topics

dmGetProperty example

The following example sets a table's Auto Append property to False if the table isn't read-only and then checks to see if the table has a one-to-many link to another table. If it does, the read-only setting of the master table is set to the same read-only setting as the detail (subject) table.

```
method UpdateProperties()


if dmGetProperty(subject.tableName, "ReadOnly") <> True then
  dmSetProperty(subject.tableName, "AutoAppend", False)
endif


if dmGetProperty(subject.tableName, "LinkType") = "One-to-many" then
  dmSetProperty(dmGetProperty(subject.tableName,"Parent"),"ReadOnly",
                dmGetProperty(subject.tableName, "ReadOnly"))
endif


endMethod
```

For information on table aliases, see Table Aliases in the Corel Paradox User's Guide help.

dmHasTable method/procedure

Reports whether a table is part of the data model of a form.

Syntax

```
dmHasTable ( const tableName String ) Logical
```

Description

dmHasTable reports whether *tableName* is a table associated with a form, where *tableName* is a valid table name or table alias.

Example

{button ,AL(`OPAL_TYPE_FORM;OPAL_METH_FODMADDTABLE;OPAL_METH_FODMREMOVETABLE;',0,"Defaultoverview",)} Related Topics

dmHasTable example

See the example for **dmAddTable** for an illustration of how to use **dmHasTable** as a procedure.

The following example shows how **dmHasTable** is used as a method. The **pushButton** method for a button named *isStockInDM* works with the form specified by the variable *thatForm*. This method opens the *Ordentry* form and then checks to see if the *Stock* table is in *thatForm's* data model. If not, the *Stock* table is added to the data model for *thatForm*.

```
; isStockInDM::pushButton
method pushButton(var eventInfo Event)
var
  thatForm Form
endVar
thatForm.load("Ordentry")                    ; open ORDENTRY form
if not thatForm.dmHasTable("stock") then     ; is Stock in data model
  msgInfo("Status", "Adding Stock to data model for form.")
  thatForm.dmAddTable("stock")               ; if not, add it
  thatForm.save()
else
  msgInfo("Status", "Stock is already in data model for form.")
endif
thatForm.close()
endMethod
```

For information on table aliases, see Table Aliases in the Corel Paradox User's Guide help.

dmLinkToFields method/procedure

Links two tables in a data model based on lists of field names.

Syntax

```
dmLinkToFields ( const masterTable String, const masterFields Array[ ] String, const
detailTable String, const detailFields Array[ ] String ) Logical
```

Description

dmLinkToFields links the tables specified in *masterTable* and *detailTable* on the field names listed in *masterFields* and *detailFields* (resizeable arrays of strings). The values of *masterTable* and *detailTable* can be table names or table aliases. The tables must already be in the form's data model.

The linking fields cannot be any of the following types: Binary, Byte, Formatted Memo, Graphic, Logical, Memo, or Object Linking and Embedding (OLE). This method returns True if successful; otherwise, it returns False. If the detail table does not have an index that matches the fields in *detailFields*, it returns False.

Examples

{button ,AL(`OPAL_TYPE_FORM;OPAL_METH_FODMLINKTOINDEX;OPAL_METH_FODMUNLINK;',0,"Default overview",)} Related Topics

dmLinkToFields examples

Example1	Linking two tables
Example2	Linking three tables

dmLinkToFields example 1

The following example creates a form, adds the Customer and Orders tables to the new specified data model, and calls **dmLinkToFields** to link the tables. It also creates some field objects and a table frame and binds them to the tables. Finally, this code runs the new form so you can see the results.

The following code specifies the names of the fields to link; you could leave this to Corel Paradox, but default linking in Corel Paradox may not give the results you expect.

```
method pushButton(var eventInfo Event)
   var
      masterTC, detailTC      TCursor
      newForm                 Form
      masterFieldsAr,
      detailFieldsAr,
      keyFieldsAr               Array[] String
      badKeyTypesAr            Array[7] String
      masterName,
      detailName,
      keyFieldName,
      newFormName             String
      newField,
      newTFrame               UIObject
      x, y, w, h, offset      LongInt
      i                       SmallInt
   endVar


   ; initialize variables
   masterName = "customer.db"
   detailName = "orders.db"
   newFormName = "custOrd.fsl"


   badKeyTypesAr[1] = "MEMO"       ; types not allowed as key fields
   badKeyTypesAr[2] = "FMTMEMO"
   badKeyTypesAr[3] = "BINARYBLOB"
   badKeyTypesAr[4] = "GRAPHIC"
   badKeyTypesAr[5] = "OLEOBJ"
   badKeyTypesAr[6] = "LOGICAL"
   badKeyTypesAr[7] = "BYTES"


   masterTC.open(masterName)
   masterTC.enumFieldNames(masterFieldsAr)


   detailTC.open(detailName)
   detailTC.enumFieldNames(detailFieldsAr)


   ; specify the key field(s)
   keyFieldName = "Customer No"


   ; make sure key field type is valid
   if badKeyTypesAr.contains(masterTC.fieldType(keyFieldName)) or
      badKeyTypesAr.contains(detailTC.fieldType(keyFieldName)) then
      msgStop("Invalid key field type:",
              keyFieldName + " in\n" +
              masterName + " or\n" + detailName)
      return
   else
      keyFieldsAr.grow(1)
      keyFieldsAr[1] = keyFieldName
   endIf


   ; create the form
   newForm.create()
   newForm.dmAddTable(masterName)
   newForm.dmAddTable(detailName)
```

```
   if newForm.dmLinkToFields(masterName, keyFieldsAr,
                             detailName, keyFieldsAr) then

   ; place objects in the form

      x = 100
      y = 100
      w = 2880
      h = 360
      offset = 10

   ; create field objects bound to master table
      for i from 1 to masterFieldsAr.size()
         newField.create(FieldTool, x, y, w, h, newForm)
         y = y + h + offset
         newField.TableName = masterName
         newField.FieldName = masterFieldsAr[i]
         newField.Visible = Yes
      endFor

   ; create a table frame bound to detail table
      newTFrame.create(TableFrameTool, x, y, w, 8 * h, newForm)
      newTFrame.TableName = detailName
      newTFrame.Visible = Yes

   ; save the form and run it
      newForm.save(newFormName)
      newForm.run()

   else

      errorShow("Link failed")
   endIf

endMethod
```

dmLinkToFields example 2

The following example shows how to use **dmLinkToFields** to link three tables 1:M:M. Like the Example 1, this code specifies which fields to link.

```
method pushButton(var eventInfo Event)
   var
      firstTable,
      secondTable,
      thirdTable      String
      firstKeyAr,
      secondKeyAr,
      thirdKeyAr      Array[] String
      newForm         Form
   endVar


   ; initialize variables
   firstTable = "customer.db"
   secondTable = "orders.db"
   thirdTable = "lineitem.db"


   firstKeyAr.grow(1)
   firstKeyAr[1] = "Customer No"
   secondKeyAr.grow(1)
   secondKeyAr[1] = "Customer No"
   ; thirdKeyAr is initialized below, after 1st link


   ; create the form
   newForm.create()


   newForm.dmAddTable(firstTable)
   newForm.dmAddTable(secondTable)
   newForm.dmAddTable(thirdTable)


   ; 1st link
   if newForm.dmLinkToFields(firstTable, firstKeyAr,
                          secondTable, secondKeyAr) then


      ; initialize arrays for 2nd link
      secondKeyAr[1] = "Order No"


      thirdKeyAr.grow(1)
      thirdKeyAr[1]  = "Order No"


      ; 2nd link
      if newForm.dmLinkToFields(secondTable, secondKeyAr,
                          thirdTable, thirdKeyAr) then


         {Code to create UIObjects in new form could go here.}


         newForm.save("ordentry.fsl")


      else
         errorShow("2:3 link failed.")
      endIf


   else
      errorShow("1:2 link failed.")
   endIf


endMethod
```

dmLinkToIndex method/procedure

Links two tables in the form's data model based on a list of field names and an index name.

Syntax

dmLinkToIndex (const ***masterTable*** String, const ***masterFields*** Array[ ] String, const ***detailTable*** String, const ***detailIndex*** String **)** Logical

Description

Links the tables specified in *masterTable* and *detailTable* to the field names listed in *masterFields* and the index specified in *detailIndex*. You can specify a Corel Paradox table's primary index by assigning an empty string to *detailIndex*.

The values of *masterTable* and *detailTable* can be table names or table aliases. The tables must already be in the form's data model. This method returns True if successful; otherwise, it returns False.

The linking fields cannot be any of the following types: Binary, Bytes, Formatted Memo, Graphic, Logical, Memo, or Object Linking and Embedding (OLE).

Example

{button ,AL(`OPAL_TYPE_FORM;OPAL_METH_FODMLINKTOFIELDS;OPAL_METH_FODMUNLINK;',0,"Defaultoverview",)} **Related Topics**

dmLinkToIndex example

The following example creates a form, adds the Customer and Orders tables to the new specified data model, and calls **dmLinkToIndex** to link the tables. It also creates some field objects and a table frame and binds them to the tables. Finally, this code runs the new form so you can see the results.

```
method pushButton(var eventInfo Event)
   var
      masterTC, detailTC      TCursor
      newForm                 Form
      masterFieldsAr,
      detailFieldsAr,
      masterKeysAr,
      detailKeysAr            Array[] String
      masterName,
      detailName,
      detailIndexName,
      newFormName             String
      newField,
      newTFrame               UIObject
      x, y, w, h, offset      LongInt
      i                       SmallInt
   endVar


   ; Initialize variables
   detailIndexName = "Customer No"
   newFormName = "idxDemo"
   masterName = "customer.db"
   detailName = "orders.db"


   masterTC.open(masterName)
   masterTC.enumFieldNames(masterFieldsAr)
   masterTC.enumFieldNamesInIndex(masterKeysAr)


   detailTC.open(detailName)
   detailTC.enumFieldNames(detailFieldsAr)



   ; create the form
   newForm.create()
   newForm.dmAddTable(masterName)
   newForm.dmAddTable(detailName)


   if newForm.dmLinkToIndex(masterName, masterKeysAr,
                            detailName, detailIndexName) then


      x = 100
      y = 100
      w = 2880
      h = 360
      offset = 10


      for i from 1 to masterFieldsAr.size()
         newField.create(FieldTool, x, y, w, h, newForm)
         y = y + h + offset
         newField.TableName = masterName
         newField.FieldName = masterFieldsAr[i]
         newField.Visible = Yes
      endFor


      newTFrame.create(TableFrameTool, x, y, w, 8 * h, newForm)
      newTFrame.TableName = detailName
      newTFrame.Visible = Yes

      newForm.save(newFormName)
```

```
      newForm.run()
   else
      errorShow("Link failed")
   endIf
endMethod
```

For information on table aliases, see Table Aliases in the Corel Paradox User's Guide help.

dmPut method/procedure

Writes data to a table in the data model.

Syntax

```
dmPut ( const tableName String, const fieldName String, const datum AnyType ) Logical
```

Description

dmPut provides access to table data in the data model. **dmPut** writes *datum* to a field in a specified table. The value of *tableName* can be a table name or a table alias. The table specified by *tableName* must be one of the tables in the data model. *fieldName* must be a field in *tableName*. This method returns True if successful; otherwise, it returns False.

Example

{button ,AL(`OPAL_TYPE_FORM;OPAL_METH_FODMGET;',0,"Defaultoverview",)} Related Topics

dmPut example

See the **dmGet** example.

dmRemoveTable method/procedure

Removes a table from the form's data model.

Syntax

```
dmRemoveTable ( const tableName String ) Logical
```

Description

dmRemoveTable removes *tableName* from a form's data model. The value of *tableName* can be a table name or a table alias. Any objects on the form that depend on the table will be undefined when the table is removed. If any UIObjects in the form are bound to the table, **dmRemoveTable** fails. It returns True if successful; otherwise, it returns False.

Example

{button ,AL(`OPAL_TYPE_FORM;OPAL_METH_FODMADDTABLE;OPAL_METH_FODMHASTABLE;',0,"Default overview",)} Related Topics

dmRemoveTable example

See the **dmAddTable** example.

For information on table aliases, see Table Aliases in the Corel Paradox User's Guide help.

dmResync method/procedure

Resynchronizes a table in the form's data model to a TCursor.

Syntax

```
dmResync ( const tableName String, var tc TCursor ) Logical
```

Description

dmResync synchronizes a specified table in a data model with the TCursor *tc*. The value of *tableName* can be a table name or a table alias.

When you resynchronize a table to a TCursor, the table's filter, index, and active record position will be changed to those of the TCursor. (For dBASE tables, the table will also take the Show Deleted setting of the TCursor.) This method works on forms in design mode or run mode.

Note

- **dmResync** only works when the TCursor is associated with the table in the data model. However, the table does not have to be displayed in the form.

Example

{button ,AL(`OPAL_TYPE_FORM;OPAL_METH_FODMATTACH;OPAL_METH_FODMHASTABLE;OPAL_METH_UIRESYNC;',0,"Defaultoverview",)} Related Topics

dmResync example

The following example shows how to use **dmResync** with the DataSource property to add items to a drop-down edit list. First, it shows how to use DataSource alone, which fills a list with values from a specified field (column) of a table. Then it shows how to use a TCursor and **dmResync** to fill a list with a specified subset of those values.

A field displayed as a drop-down edit list is a compound object: the field object (which displays the field value) contains a list object (which contains the items in the list). In a form, the list object is represented by the down-arrow (the arrow you click to display the list).

The usual place to attach list-building code is the list object's built-in **open** method, but you can attach the code to other methods or even to other objects (as shown in the second part of this example).

Assume a form contains a field object displayed as a drop-down edit list. The field object is bound to the ShipVia field of the *Orders* table. The following code is attached to the built-in **open** method of the list object (not the field object) named *shipViaList*. It fills the list with all the values in the ShippingCo field of the *Shippers* table in the working directory.

```
; shipViaList::open
; Full containership path: form.page.ShipVia.shipViaList
method open (var eventInfo Event)
   doDefault
   ; Fills list with all values in ShippingCo field of Shippers table.
   self.DataSource = "[Shippers.ShippingCo]"
endMethod
```

The following code uses **dmResync** to filter the list based on the value of another field. The premise here is that certain shipping methods are less expensive (and so more desirable) in certain parts of the country. When the user changes the value of the *State* field, this code updates the items in the list of shippers.

```
; State::changeValue
method changeValue (var eventInfo ValueEvent)
   var
      tcShippers   TCursor
      stStateCode,
      stFldName,
      stDmTbName   String
      dyCriteria   DynArray[] AnyType
   endVar


   doDefault ; Execute the built-in code to commit the field value.
   if eventInfo.errorCode() <> 0 then
      return ; If there's an error, exit the method.
   endIf


   stStateCode = self.Value  ; Get the value of the State field.
   stFldName   = "State"     ; Filter on the State field.
   stDmTbName  = "Shippers"


   dyCriteria[stFldName] = stStateCode


   ; Associate a TCursor with a table in the form's data model.
   dmAttach(tcShippers, stDmTbName)


   tcShippers.setGenFilter(dyCriteria) ; Set a filter on the TCursor.
   ; You could also set an index, etc.


   ; Synchronize the table in the data model with the TCursor.
   ; The table takes the filter from the TCursor.
   dmResync(stDmTbName, tcShippers)


   ; Now the list displays only the shippers for the specified state.
   ShipVia.shipViaList.DataSource = "[Shippers.ShippingCo]"


endMethod
```

For information on table aliases, see Table Aliases in the Corel Paradox User's Guide help.

dmSetProperty method/procedure

Sets the value of a specified table property.

Syntax

```
dmSetProperty ( const tableName String, const propertyName String, value AnyType) Logical
```

Description

dmSetProperty lets you change the value of a property (specified in *propertyName*) associated with the table specified in *tableName* and found in the data model.

The value of *tableName* can be a table name or a table alias. The value of *propertyName* is one of the following properties:

AutoAppend	Set *propertyValue* to True to set AUTO APPEND ON for the table; otherwise, set it to False.
Name	The value of *propertyValue* specifies the table's alias as a string. The operation fails if the table alias is already in use.
ReadOnly	Set *propertyValue* to True if READONLY should be True for the table; otherwise, set it to False.
StrictTranslation	Set *propertyValue* to True if STRICT TRANSLATION should be True for the table.; otherwise, set it to False.
Touched	Set *propertyValue* to True when the user has made changes not yet committed.

Example

{button ,AL(`OPAL_TYPE_FORM;OPAL_METH_FODMGETPROPERTY;',0,"Defaultoverview",)} Related Topics

dmSetProperty example

See the **dmGetProperty** example.

For information on table aliases, see Table Aliases in the Corel Paradox User's Guide help.

dmUnlink method/procedure

Unlinks two tables in the form's data model.

Syntax

```
dmUnlink ( const masterTable String, const detailTable String ) Logical
```

Description

dmUnlink breaks the link between the tables specified in *masterTable* and *detailTable*. *masterTable* must refer to the master table in the link, and *detailTable* must refer to the detail table in the link. The values of *masterTable* and *detailTable* can be table names or table aliases.

This method fails if the tables are not in the data model; it also fails if they are in the data model but not linked.

This method returns True if successful; otherwise, it returns False.

Example

{button ,AL(`OPAL_TYPE_FORM;OPAL_METH_FODMLINKTOFIELDS;OPAL_METH_FODMLINKTOINDEX;',0,"Defaultoverview",)} Related Topics

dmUnlink example

The following example uses **dmUnlink** to break the link between two tables:

```
method pushButton(var eventInfo Event)

   var
      theForm            Form
      masterTable,
      oldDetailTable,
      newDetailTable,
      oldFormName,
      newFormName         String
      newKeysAr         Array[] String
   endVar

   ; initialize variables
   oldFormName = "custOrd"
   newFormName = "newOrd"

   masterTable    = "CUSTOMER"
   oldDetailTable = "ORDERS"
   newDetailTable = "NEW_ord"

   newKeysAr.grow(1)
   newKeysAr[1] = "Customer No"

   ; load the form and change the data model
   theForm.load(oldFormName)

   if theForm.dmHasTable(masterTable) and
      theForm.dmHasTable(oldDetailTable) then

      theForm.dmAddTable(newDetailTable)
      theForm.dmUnlink(masterTable, oldDetailTable)

      theForm.dmLinkToFields(masterTable, newKeysAr,
                             newDetailTable, newKeysAr)

      theForm.ORDERS.TableName = newDetailTable

      theForm.dmRemoveTable(oldDetailTable)
      theForm.save(newFormName)

   else
      errorShow()
   endIf

endMethod
```

For information on table aliases, see Table Aliases in the Corel Paradox User's Guide help.

enumDataModel method/procedure

Lists the tables in the form's data model.

Syntax

enumDataModel (const ***tableName*** String **)** Logical

Description

enumDataModel creates a table that lists information about the tables in the form's data model. Use the argument *tableName* to specify a name for the table. If *tableName* already exists, this method overwrites it without asking for confirmation. If *tableName* is already open, this method fails. You can include an alias or path in *tableName*; if no alias or path is specified, Corel Paradox creates *tableName* in the working directory (:WORK:).

The structure of the created table is

Field Name	Type	Description
TableName	A128	Table alias, if it exists, or filename of the table (without file extension)
PropertyName	A64	A property name
PropertyValue	A255	Value of the corresponding property

Example

{button ,AL(`OPAL_TYPE_FORM;OPAL_METH_FODMENUMLINKFIELDS;OPAL_METH_FODMGETPROPERTY;OPAL_METH_FOENUMSOURCE;',0,"Defaultoverview",)} Related Topics

enumDataModel example

In the following example, a form contains a button named *enumerateDataModel*. The **pushButton** method for *enumerateDataModel* uses **enumDataModel** as a procedure to enumerate the properties of all the tables in the data model for the current form to a table called DMORDERS.DB. The method then opens a Table window for the *DMOrders* table.

```
;enumerateDataModel::pushButton
method pushButton(var eventInfo Event)
   var
      tv   TableView
   endVar


   enumDataModel("dmOrders.db")
   tv.open("dmOrders.db")
endMethod
```

Property Names for enumDataModel (Form type)

Property	Description
AutoAppend	Returns True if AUTO APPEND is set to True for the table; otherwise, it returns False
FullName	Returns the full filename (including path or alias) of the table
Index	Returns the name of the index (as a string) that is currently used to view the table. For a child table, it returns the name of the index chosen in the link diagram. For a master table or unlinked table, it returns the setting of ORDER/RANGE. It returns an empty string when the primary key is used.
LinkFields	Returns a comma-separated list of fields that define the link. If the detail table is a dBASE table and uses an expression index, the expression is returned in angled brackets. Examples: <FIRSTNAME + LASTNAME> means an expression index based on the fields named FIRSTNAME and LASTNAME; <FIRSTNAME + LASTNAME;QTY > 1> means an expression index based on the fields named FIRSTNAME and LASTNAME with QTY > 1 as a subset condition.
LinkType	Returns a string describing the way the table relates to its master: None, One-to-one, or One-to-many
Name	Returns the table's alias (as a string) if it exists; otherwise, returns an empty string
Next	Returns the name (as a string) of the next object in the same container
One-to-many	Returns the name (as a string) of the first detail table linked 1:M to this table
One-to-one	Returns the name (as a string) of the first detail table linked 1:1 to this table
Parent	Returns the table name (as a string) of this table's master in the data model. For example, in a CUSTOMER->>BOOKORD form, dmGetProperty("BOOKORD,""PARENT") = "CUSTOMER.DB." If the table has no master, an empty string is returned.
Read-only	Returns True if READONLY is set to True for the table; otherwise, it returns False
StrictTranslation	Returns True if STRICT TRANSLATION is set to True for the table; otherwise, it returns False
TagName	Returns the tag name (as a string) for the current dBASE index (if any); otherwise, it returns an empty string

enumSource method/procedure

Creates a table that lists the methods for each object in a form.

Syntax

```
enumSource ( const tableName String [ , const recurse Logical ] ) Logical
```

Description

enumSource creates a Corel Paradox table that lists every object for which you have written a method, and the ObjectPAL source code for the method. Use the argument *tableName* to specify a name for the table. If *tableName* already exists, this method overwrites it without asking for confirmation. If *tableName* is already open, this method fails. You can include an alias or path in *tableName*; if no alias or path is specified, Corel Paradox creates *tableName* in the working directory.

The structure of the created table is as follows:

Field name	Type	Size
Object	A	128
MethodName	A	128
Source	M	64

The Object field contains the full path name of the object.

If *recurse* is False, this method returns only the method definitions for the form. To include the source code of methods for all objects contained by the form, *recurse* must be True.

Example

{button ,AL(`OPAL_TYPE_FORM;OPAL_METH_FOENUMSOURCETOFILE;OPAL_METH_FOENNAM;OPAL_METH_FOENPRO;',0,"Defaultoverview",)} Related Topics

enumSource example

In the following example, a form contains a button named *getSource*. The **pushButton** method for *getSource* uses **enumSource** as a procedure to enumerate the source code for the current form to a table named TEMPSORC.DB. The method opens a Table window for the *Tempsorc* table and waits for the user to close it. The method then opens the *Sitenote* form to *siteForm*, uses enumSource as a method to write the source code for *siteForm* to a table named SITESORC.DB, and views the table:

```
; getSource::pushButton
method pushButton(var eventInfo Event)
var
  siteForm   Form
  tempTable  TableView
endVar


siteForm.open("Sitenote.fsl")        ; open another form


; write source for siteForm to SITESORC.DB
siteForm.enumSource("sitesorc.db", True)
siteForm.close()                ; close the form
tempTable.open("sitesorc.db")   ; view the new table
tempTable.wait()                ; wait for the user to close
                                ; the table
endMethod
```

enumSourceToFile method/procedure

Creates a file that lists the methods for each object in a form.

Syntax

```
enumSourceToFile ( const fileName String [ , const recurse Logical ] ) Logical
```

Description

enumSourceToFile creates a text file that lists every object for which you've written a method, and the ObjectPAL source code for the method. Use the argument *fileName* to specify a name for the file. If *fileName* already exists, this method overwrites it without asking for confirmation. You can include an alias or path in *fileName*; if no alias or path is specified, Corel Paradox creates *fileName* in the working directory.

If *recurse* is False, this method returns only the method definitions for the form. To include the source code of methods for all objects contained by the form, *recurse* must be True.

Example

enumSourceToFile example

In the following example, code is attached to the **pushButton** method for a button named *getSourceToFile*. This method writes all the source code for the current form to TEMPSORC.TXT. The method then opens the *Sitenote* form and writes all the code for that form to a file named SITESORC.TXT:

```
; getSourceToFile::pushButton
method pushButton(var eventInfo Event)
var
  siteForm   Form
endVar
enumSourceToFile("tempsorc.txt", True) ; writes all source for the
                                       ; current form to TEMPSORC.TXT


siteForm.open("Sitenote.fsl")              ; open another form
; write source for siteForm to SITESORC.TXT
siteForm.enumSourceToFile("sitesorc.txt", True)
siteForm.close()                       ; close the form
endMethod
```

enumTableLinks method/procedure

Creates a table that lists the tables linked in a form.

Syntax

```
enumTableLinks ( const tableName String ) Logical
```

Description

enumTableLinks creates a Corel Paradox table that lists the names of tables linked in a form and the types of links. Use the argument *tableName* to specify a name for the table. If *tableName* already exists, this method overwrites it without asking for confirmation. If *tableName* is already open, this method fails. You can include an alias or path in *tableName*; if no alias or path is specified, Corel Paradox creates *tableName* in the working directory.

This method creates a table that contains one record for each table in the data model. The structure of the table is:

Field name	Type	Description
Table	A255*	Table name, without alias, path, or extension (for example, ORDERS).
Parent	A255*	Name of parent table, or blank if table has no parent.
LinkType	A24*	Type of link between table and master table: None, One-to-many, or One-to-one.

Example

{button ,AL(`OPAL_TYPE_FORM;OPAL_METH_FOENUMDATAMODEL;OPAL_METH_FOENNAM;OPAL_METH_FOENPRO;OPAL_METH_TCENUMREFINTSTRUCT;',0,"Defaultoverview",)} Related Topics

enumTableLinks example

In the following example, the **pushButton** method for a button named *showTableLinks* writes table links for the current form to a table named TEMPLINK.DB. The method then opens the *Sitenote* form and writes the table links for that form to a table named SITENOTE.DB.

```
; showTableLinks::pushButton
method pushButton(var eventInfo Event)
var
  siteForm  Form
  tempTable TableView
endVar
enumTableLinks("templink.db")          ; lists links to current form
tempTable.open("templink")
tempTable.wait()
siteForm.open("Sitenote.fsl")
siteForm.enumTableLinks("Sitenote.db") ; lists links to siteForm
siteForm.close()
tempTable.open("Sitenote.db")
tempTable.wait()
tempTable.close()
endMethod
```

enumUIObjectNames method

Creates a table that lists the UIObjects contained in a form.

Syntax

enumUIObjectNames (const ***tableName*** String) Logical

Description

enumUIObjectNames creates a Corel Paradox table that lists the name and type of each object contained in a form. Use the argument *tableName* to specify a name for the table. If *tableName* already exists, this method overwrites it without asking for confirmation. If *tableName* is already open, this method fails. You can include an alias or path in *tableName*; if no alias or path is specified, Corel Paradox creates *tableName* in the working directory.

The structure of tableName is as follows:

Field Name	Type	Size
ObjectName	A	128
ObjectClass	A	32

Note

- ObjectName includes the entire path name of the object.

Example

{button ,AL(`OPAL_TYPE_FORM;OPAL_METH_FOENPRO;OPAL_METH_FOENUMSOURCE;OPAL_METH_FOENUMSOURCETOFILE;',0,"Defaultoverview",)} **Related Topics**

enumUIObjectNames example

In the following example, the **pushButton** method for a button named *getObjectNames* opens the *Sitenote* form and enumerates all the object names on the form to a table named *Siteobjs*. The method then opens the *Siteobjs* table and waits for the user to close the table.

```
; getObjectNames::pushButton
method pushButton(var eventInfo Event)
var
  siteForm  Form
  tempTable TableView
endVar
if siteForm.open("Sitenote.fsl") then           ; open the form
  siteForm.enumUIObjectNames("siteobjs.db") ; write object names
                                            ; SITEOBJS.DB
  siteForm.close()                          ; close the form
  tempTable.open("siteobjs")                ; open the new table
  tempTable.wait()                          ; wait for return
  tempTable.close()                         ; close after return
endIf
endMethod
```

enumUIObjectProperties method

Lists the properties of each UIObject contained in a form.

Syntax

```
enumUIObjectProperties ( const tableName String ) Logical
```

Description

enumUIObjectProperties creates a Corel Paradox table that lists the name, property name, and property value of each object contained in a form. Use the argument *tableName* to specify a name for the table. If *tableName* already exists, this method overwrites it without asking for confirmation. If *tableName* is already open, this method fails.

The structure of *tableName* is:

Field name	Type	Size
ObjectName	A	128
PropertyName	A	64
PropertyType	A	48
PropertyValue	A	255

Example

{button ,AL(`OPAL_TYPE_FORM;OPAL_METH_UIEOPT;',0,"Defaultoverview",)} Related Topics

enumUIObjectProperties example

In the following example, the **pushButton** method for a button named *getProps* writes the properties for all objects contained by the current form to a table named *Tempprop*:

```
; getProperties::pushButton
method pushButton(var eventInfo Event)
var
  siteForm  Form
  tempTable TableView
endVar
if siteForm.open("Sitenote.fsl") then
  message("Enumerating properties to Siteprop table.")
  siteForm.enumUIObjectProperties("siteProp.db")
  tempTable.open("siteprop")
  message("Close the table to continue.")
  tempTable.wait()
  tempTable.close()
endIf
; to enumerate objects for current form, use the UIObject
; type method enumUIObjectProperties
; thisForm is the object ID for current form
message("Enumerating properties to Tempprop table.")
   enumUIObjectProperties("tempprop.db")
tempTable.open("tempprop")
message("Close the table to continue.")
tempTable.wait()
tempTable.close()
endMethod
```

formCaller procedure

Creates a handle to the calling form.

Syntax

```
formCaller ( var caller Form ) Logical
```

Description

formCaller assigns the handle of the current form's calling form to *caller*, if the form is waiting. If the current form was not opened by another form, and the form that opened the current form is not waiting for the current form, the method returns False and *caller* is unassigned.

Example

{button ,AL(`OPAL_TYPE_FORM;OPAL_METH_FOWAIT;',0,"Defaultoverview",)} **Related Topics**

formCaller example

In the following example, the **pushButton** method for *whoCalledMe* finds out which form called the current form:

```
; callOtherForm::pushButton  (calling form)
method pushButton(var eventInfo Event)
var
  siteForm  Form
endVar
siteForm.open("sitenote.fsl")  ; open siteForm
siteForm.wait()                ; wait for siteForm to return
siteForm.close()               ; close siteForm
endMethod
```

The following code is for *whoCalledMe* on the current form.

```
; whoCalledMe::pushButton
method pushButton(var eventInfo Event)
var
  myCaller     Form
  callerTitle  AnyType
endVar
if formCaller(myCaller) then        ; try to get a handle to
                                    ; the calling form
  callerTitle = myCaller.getTitle() ; get the form's title
  msgInfo("FYI", "I was called by: \n" + callerTitle)
endif
formReturn()
endMethod
```

formReturn procedure

Returns control to a suspended method.

Syntax

```
formReturn ( [ const returnValue AnyType ] )
```

Description

When one form opens another form and calls **wait**, the first form suspends ObjectPAL execution (in effect, yielding to the second form) until the second form returns control by calling **formReturn**. You can choose to return a value to the first form in *returnValue*. You can also use **formReturn** to return control (and a value) from a script.

formReturn posts a message to the Windows message queue; therefore, ObjectPAL statements that follow **formReturn** will execute before the form returns control.

If no other form is waiting for the current form, **formReturn** closes the current form. If a form is waiting for the current form, **formReturn** does not close the current form.

Example

{button ,AL(`OPAL_TYPE_FORM;OPAL_METH_FOFORMCALLER;OPAL_METH_FOWAIT;OPAL_METH_SCRUN;',0,"Defaultoverview",)} **Related Topics**

formReturn example

The following example consists of three methods. The **pushButton** method for *openDialog* opens another form as a dialog box and waits for it to return a value. The other two methods are attached to buttons in the dialog box form. They use **formReturn** to return control and values to the calling form. Note that the calling form must call **close** to close the dialog box; the call to **formReturn** does not close the dialog box.

```
; openDialog::pushButton
method pushButton(var eventInfo Event)
var
  dlgForm     Form
  whichButton String
endVar
if dlgForm.openAsDialog("foforet2", WinStyleDefault,
                        1440, 1440, 7200, 5760) then
  ; waits until dlgForm calls formReturn or is closed
  ; returned value is stored to whichButton
  whichButton = String(dlgForm.wait())
  dlgForm.close()
  ; return value is cast as a String so that it will be correct
  ; type even if user closes dialog box from the system menu
  msgInfo("Button pressed", whichButton)
else
  msgStop("Stop", "Couldn't open the form.")
endIf
endMethod
```

The following method is attached to the **pushButton** method for *OKButton* in *dlgForm.* It returns a value of OK when it returns control to the method that called wait:

```
; OKButton::pushButton
method pushButton(var eventInfo Event)
formReturn("OK")      ; return "OK" to calling form
endMethod
```

The following method is attached to *cancelButton* in *dlgForm.* It returns a value of Cancel when it returns control to the method that called **wait**. The **message** statement that follows the call to **formReturn** is not required; it is included here to show that statements following a call to **formReturn** execute before control is returned to the calling form.

```
; cancelButton::pushButton
method pushButton(var eventInfo Event)
formReturn("Cancel")  ; return "Cancel" to calling form
message("Cancel")     ; This statement will execute.
endMethod
```

getFileName method/procedure

Returns the path, filename, and extension of the associated form.

Syntax

getFileName( ) String

Description

As a method, **getFileName** returns the path, filename, and extension of the form associated with a Form variable. As a procedure, it returns the path, filename, and extension of the current form. Compare this method to **getTitle**, which returns the text in a Form window's Title Bar.

Example

{button ,AL(`OPAL_TYPE_FORM;OPAL_METH_FOGETT;',0,"Defaultoverview",)} **Related Topics**

getFileName example

The following example displays the filename of the current form in the Status Bar.

```
method pushButton(var eventInfo Event)
   message(getFileName())
endMethod
```

getPosition method/procedure

Reports the position of a window onscreen.

Syntax

```
getPosition ( var x LongInt, var y LongInt, var w LongInt, var h LongInt )
```

Description

getPosition gets the position of a window relative to the Corel Paradox desktop. The arguments *x* and *y* contain the horizontal and vertical coordinates of the upper-left corner of the form (in twips), and *w* and *h* contain the width and height (in twips).

To ObjectPAL, the screen is a two-dimensional grid, with the origin (0, 0) at the upper-left corner of an object's container, positive x-values extending to the right, and positive y-values extending down.

For dialog boxes, and for the Corel Paradox desktop application, the position is given relative to the entire screen; for forms, reports, and Table windows, the position is given relative to the Corel Paradox desktop.

Example

**{button ,AL(`OPAL_TYPE_FROM;OPAL_TYPE_APPLICATION;OPAL_METH_FOSPOS;;;',0,"Defaultoverview",)
} Related Topics**

getPosition example

In the following example, the **pushButton** method for *moveOtherForm* opens a form and gets its position. The method then opens a second instance of the same form and sets its position so that no part of the second form overlaps the first.

```
; moveOtherForm::pushButton
method pushButton(var eventInfo Event)
var
  siteFormOne,
  siteFormTwo    Form
  x, y, w, h     LongInt
endVar
if siteFormOne.open("Sitenote") then
  siteFormOne.getPosition(x, y, w, h)
  siteFormTwo.open("Sitenote.fsl")    ; open another instance
  ; set position so that no part overlaps other instance
  siteFormTwo.setPosition(x + w, y + h, w, h)
endif
endMethod
```

getProtoProperty method/procedure

Reports the value of a specified property of a prototype object.

Syntax

```
getProtoProperty ( const objectType SmallInt, propertyName String ) AnyType
```

Description

getProtoProperty returns the value of the property specified in *propertyName* of the prototype object specified in *objectType*. To specify *objectType*, use one of the UIObjectTypes constants. If called as a method, **getProtoProperty** operates on prototype objects in the style sheet of the specified form. If called as a procedure, **getProtoProperty** uses the style sheet of the current form.

Example

{button ,AL(`OPAL_TYPE_FORM;OPAL_METH_FOSETPROTOPROPERTY;OPAL_METH_FOGETSTYLESHEET;OPAL_METH_FOSAVESTYLESHEET;',0,"Defaultoverview",)} Related Topics

getProtoProperty example

The following example uses **getProtoProperty** to store the current default color for the box tool. Next, it specifies a new box color and creates three new boxes, and then restores the default box color.

```
   const
      kOneInch = 1440 ; One inch = 1,440 twips.
   endConst
method mouseClick(var eventInfo MouseEvent)
   var
      uiRedBox       UIObject
      thisForm      Form
      liDefaultBoxColor    LongInt
   endVar
   thisForm.attach() ; Get a handle to this form.


   ; Get current default color.
   liDefaultBoxColor = thisForm.getProtoProperty(BoxTool, "Color")


   ; Set box color and create 3 boxes using new prototype.
   thisForm.setProtoProperty(BoxTool, "Color", Red)
   uiRedBox.create(BoxTool, kOneInch, kOneInch, kOneInch, kOneInch)
   uiRedBox.Visible = Yes
   uiRedBox.create(BoxTool, 2 * kOneInch, kOneInch, kOneInch, kOneInch)
   uiRedBox.Visible = Yes
   uiRedBox.create(BoxTool, 3 * kOneInch, kOneInch, kOneInch, kOneInch)
   uiRedBox.Visible = Yes


   ; Restore the default box color.
   thisForm.setProtoProperty(BoxTool, "Color", liDefaultBoxColor)
endMethod
```

getSelectedObjects method/procedure

Creates an array that lists the selected objects in a form.

Syntax

```
getSelectedObjects ( var objects Array[ ] UIObject ) SmallInt
```

Description

getSelectedObjects creates an array *objects* that lists the selected objects of a form and returns the number of objects selected. This procedure is useful for creating routines that manipulate objects on forms in design mode.

Example

{button ,AL(`OPAL_TYPE_FORM;OPAL_METH_FOSETSELECTEDOBJECTS;',0,"Defaultoverview",)} Related Topics

getSelectedObjects example

The following example creates a form that contains three boxes, selects two of the boxes, displays their names in a dialog box, and sets their color to blue:

```
;btnObjectsSelected :: pushButton
const
   kOneInch = 1440 ; One inch = 1,440 twips.
endConst


method pushButton(var eventInfo Event)
   var
      foTemp   Form
      arObjects   Array[] UIObject
      arObjNames   Array[] String
      uiVar      UIObject
      si,
      siSelObj   SmallInt
      stBoxName   String

   endVar


   foTemp.create()


   ; Create 3 boxes.
   for si from 1 to 3
      uiVar.create(BoxTool, si * kOneInch, si*kOneInch,
                   kOneInch, kOneInch, foTemp)
      uiVar.Name = "Box" + String(si)
      uiVar.Visible = Yes
   endFor


   ; Select Box2 and Box3 by setting the Select property.
   for si from 2 to 3
      stBoxName = "Box" + String(si)
      uiVar.attach(foTemp.(stBoxName))
      uiVar.Select = Yes
   endFor


   ; Get the selected objects.
   siSelObj = foTemp.getSelectedObjects(arObjects)
   siSelObj.view("Number of selected objects:")


   ; Get the names of the selected objects.
   arObjNames.setSize(siSelObj)
   for si from 1 to siSelObj
      uiVar.attach(arObjects[si])
      arObjNames[si] = uiVar.Name
   endFor
   arObjNames.view("Names of selected objects:")


   ; Change the color of the selected objects.
   for si from 1 to arObjects.size()
      uiVar.attach(arObjects[si])
      uiVar.Color = Blue
   endFor


   foTemp.close()
endMethod
```

getStyleSheet method/procedure

Returns the name of a form's style sheet.

Syntax

getStyleSheet ( ) String

Description

getStyleSheet returns the filename of a form's style sheet. If the style sheet is in the working directory (:WORK:), **getStyleSheet** returns the filename and extension, if any (e.g., COREL.FT); otherwise, **getStyleSheet** returns the full path (e.g., C:\COREL\SUITE8\PARADOX\COREL.FT).

If called as a method, **getStyleSheet** returns the filename of the style sheet of the specified form. If called as a procedure, it uses the style sheet of the current form.

getStyleSheet returns the name of the style sheet used by the specified form, which may be different from the Corel Paradox system style sheet. To get the name of the default screen style sheet, call the **getDefaultScreenStyleSheet** procedure defined for the System type. To get the name of the default printer style sheet, call the **getDefaultPrinterStyleSheet** procedure defined for the System type.

Example

{button ,AL(`OPAL_TYPE_FORM;OPAL_METH_FOSETSTYLESHEET;OPAL_METH_FOSAVESTYLESHEET;OPAL_METH_FOGETPROTOPROPERTY;OPAL_METH_FOSETPROTOPROPERTY;OPAL_METH_SYGETDEFAULTPRINTERSTYLESHEET;OPAL_METH_SYGETDEFAULTSCREENSTYLESHEET;',0,"Defaultoverview",)} **Related Topics**

getStyleSheet example

See the **setStyleSheet** example.

getTitle method/procedure

Returns the text in the window's Title Bar.

Syntax

getTitle ( ) String

Description

getTitle returns the text in the Title Bar of the window that contains the object.

Example

{button ,AL(`OPAL_TYPE_FORM;OPAL_METH_FOSTIT;OPAL_METH_FOATTA;OPAL_TYPE_APPLICATION;',0,"Defaultoverview",)} Related Topics

getTitle example

In the following example, the **pushButton** method for *showTitle* opens a form, gets the new form's title, and displays the title in a dialog box. This method then switches the open form to the Form Design window and retrieves its title again.

```
; showTitle::pushButton
method pushButton(var eventInfo Event)
var
   siteForm  Form
   titleText String
endVar
siteForm.open("Sitenote.fsl")
titleText = siteForm.getTitle() ; reads window title into titleText
msgInfo("Title:", titleText)    ; displays "Form : SITENOTE.FSL"
siteForm.design()               ; switch to the Form Design window
sleep()                         ; yield!
titleText = siteForm.getTitle() ; get the Form Design window title
msgInfo("Title:", titleText)    ; displays "Form Design: SITENOTE.FSL"
siteForm.close()
endMethod
```

hide method/procedure

Makes a window invisible.

Syntax

```
hide ( )
```

Description

hide makes a window invisible but doesn't close the window.

Example

{button ,AL(`OPAL_TYPE_FORM;OPAL_METH_FOSHOW;OPAL_METH_FOBRTT;OPAL_METH_FOOPEN;OPAL_METH_FOOPAD;OPAL_TYPE_APPLICATION;',0,"Defaultoverview",)} Related Topics

hide example

In the following example, the **pushButton** method for *hideForm* opens a form, hides it and then shows it again:

```
; hideForm::pushButton
method pushButton(var eventInfo Event)
var
  siteForm Form
endVar
siteForm.open("Sitenote.fsl")         ; displays Sitenote form
siteForm.hide()                   ; makes form invisible
siteForm.action(DataEnd)          ; move to the end of the table
siteForm.action(DataBeginEdit)    ; start edit mode
siteForm.action(DataInsertRecord) ; insert a new, blank record
if NOT siteForm.isVisible() then
  msgInfo("Status", "It's hidden.")
endif
message("Come out, come out, wherever you are!")
siteForm.show()                   ; make form visible again
if siteForm.isVisible() then
  msgInfo("Status", "It's visible.")
endif
endMethod
```

hideToolbar procedure

Makes the standard Toolbar invisible.

Syntax

```
hideToolbar ( )
```

Description

hideToolbar removes the standard Toolbar from the desktop. You must call **showToolbar** to restore the Toolbar.

Example

{button ,AL(`OPAL_TYPE_FORM;OPAL_METH_FOISBARSHO;OPAL_METH_FOSHOBAR;',0,"Defaultoverview",)} **Related Topics**

hideToolbar example

In the following example, the **pushButton** method for the *toggleToolbar* button checks whether the Toolbar is showing. If the Toolbar is visible, this method hides it; if the Toolbar isn't visible, this method shows it:

```
; toggleToolbar::pushButton
method pushButton(var eventInfo Event)
if isToolbarShowing() then   ; if Toolbar is off
   hideToolbar()             ; hide it
else                         ; otherwise
   showToolbar()             ; show it
endif
endMethod
```

isCompileWithDebug method

Reports the status of the Compile With Debug setting.

Syntax

```
isCompileWithDebug ( ) Logical
```

Description

isCompileWithDebug reports the status of the Compile With Debug setting that can be set interactively during form design. **isCompileWithDebug** returns True if Compile With Debug is set in the form; otherwise, it returns False.

Example

isCompileWithDebug example

In the following example, the central form of a management system has two buttons: *getCompileStatus* and *setCompileStatus*. The pushButton method of each button opens the Windows 95 Explorer dialog to allow a user to select the file that will be examined/manipulated. Each method analyzes the *fileName* selected to determine the file Type and to open the file under the appropriate object type.

The following code is attached to the pushButton method for *getCompileStatus*:

```
; getCompileStatus::pushButton
method pushButton(var eventInfo Event)
var
   theForm  Form         ;// Object variable for forms
   theLibrary  Library      ;// Object variable for libraries
   theScript  Script      ;// Object variable for scripts
   fbi  FileBrowserInfo      ;// File Browser information structure
   selectedFile  String      ;// FileName selected by user
   fileType  String      ;// File type of file selected by user
   status  Logical      ;// Debug status of the selected file
endVar
         ;//Set allowable file types: Forms, Libraries, and Scripts
   fbi.AllowableTypes = fbForm + fbLibrary + fbScript
   if fileBrowser(selectedFile, fbi) then
      ;// The user selected a file
     fileType = upper(substr(selectedFile, selectedFile.size() - 2, 3))
      switch
         case fileType = "FSL" :
            ;// Load the Form
            theform.load(fbi.Drive + fbi.Path + selectedFile)
            ;// Determine its status
            status = theForm.isCompileWithDebug()
            ;// Close the Form
            theForm.close()

         case fileType = "LSL" :
            ;// Load the Library
            theLibrary.load(fbi.Drive + fbi.Path + selectedFile)
            ;// Determine its status
            status = theLibrary.isCompileWithDebug()
            ;// Close the Library
            theLibrary.close()

         case fileType = "SSL" :
            ;// Load the Script
            theScript.load(fbi.Drive + fbi.Path + selectedFile)
            ;// Determine its status
            status = theScript.isCompileWithDebug()
            ;// Close the Script
            theScript.close()
      endSwitch
      ;// Inform the user
      msgInfo(selectedFile + " compiled with Debug information?", status)
   else
      ;// The user didn't select a file
      msgInfo("No file selected", "Please try again.")
   endIf
endMethod
```

isDesign method/procedure

Reports whether a form is displayed in a Form Design window.

Syntax

isDesign ( ) Logical

Description

isDesign returns True if a form is displayed in a Form Design window; otherwise, it returns False.

Example

{button ,AL(`OPAL_TYPE_FORM;OPAL_METH_FOIMAX;OPAL_METH_FOIMIN;',0,"Defaultoverview",)}

Related Topics

isDesign example

In the following example, **enumFormNames** is used to populate an array *ar* with the names of the open forms. A **for** loop then steps through the array and saves the form if it is in design mode.

```
;btnSaveForms :: pushButton
method pushButton(var eventInfo Event)
   var
      ar               Array[] AnyType
      siCounter   SmallInt
      f                  Form
   endVar


   enumFormNames(ar)


   for siCounter from 1 to ar.size()
      f.attach(ar[siCounter])
      if f.getFileName() = "" then
         msgStop("Warning", "At least one form is a new form.")
      else
         if f.isDesign() then
            f.save()
         endIf
      endIf
   endFor
endMethod
```

isMaximized method/procedure

Reports whether a window is displayed at its maximum size.

Syntax

isMaximized ( ) Logical

Description

isMaximized returns True if a form is displayed full screen; otherwise, it returns False.

Example

{button ,AL(`OPAL_TYPE_FORM;OPAL_TYPE_FOMAXI;OPAL_TYPE_FOMINI;OPAL_TYPE_FOIMIN;OPAL_TYPE_APPLICATION;',0,"Defaultoverview",)} Related Topics

isMaximized example

In the following example, the **pushButton** method for the *cycleSize* button (on the current form) opens or attaches to the *Sitenote* form with the variable *siteForm*. If *siteForm* is maximized, this method minimizes it. If *siteForm* is minimized, this method restores it to its previous size with the **show** method. If *siteForm* is neither maximized nor minimized, this method maximizes it:

```
; cycleSize::pushButton
method pushButton(var eventInfo Event)
var
  siteForm Form
endVar
; try attaching to form, since it might be open
if NOT siteForm.attach("Form : SITENOTE.FSL") then
  ; if attaching fails, try opening the form
  if NOT siteForm.open("sitenote.fsl") then
    msgStop("Failed", "Couldn't open Sitenote.")
    return     ; if open fails, give up
  endif
endif


; if we reach this point, we have a good form handle
switch
  case isMaximized()  :                   ; if forms are maximized
    msgInfo("Status", "Siteform is maximized.")
    siteForm.show()                       ; restore size
  case siteForm.isMinimized() :           ; if form is minimized
    msgInfo("Status", "Siteform is minimized.")
    siteForm.maximize()
  case NOT (siteForm.isMaximized() OR siteForm.isMinimized()):
    msgInfo("Status", "Siteform is neither minimized or maximized.")
    siteForm.minimize()                           ; minimize
  otherwise :
    msgStop("Stop", "Unable to change size of Siteform.")
endswitch
endMethod
```

isMinimized method/procedure

Reports whether a window is displayed as an icon.

Syntax

isMinimized ( ) Logical

Description

isMinimized returns True if a form is displayed as an icon; otherwise, it returns False.

Example

{button ,AL(`OPAL_TYPE_FORM;OPAL_METH_FOMINI;OPAL_METH_FOIMAX;OPAL_METH_FOMAXI;OPAL_TYPE_APPLICATION;',0,"Defaultoverview",)} Related Topics

isMinimized example

See the **isMaximized** example.

isToolbarShowing procedure

Reports whether the standard Toolbar is visible.

Syntax

isToolbarShowing ( ) Logical

Description

isToolbarShowing returns True if the standard Toolbar is visible; otherwise, it returns False.

Example

{button ,AL(`OPAL_TYPE_FORM;OPAL_METH_FOHTOOLBAR;OPAL_METH_FOSHOBAR;',0,"Defaultoverview",)} Related Topics

isToolbarShowing example

See the **hideToolbar** example.

isVisible method/procedure

Reports whether any part of a window is displayed.

Syntax

isVisible ( ) Logical

Description

isVisible returns True if any part of a window is displayed (not hidden); otherwise, it returns False.

Example

{button ,AL(`OPAL_TYPE_FORM;OPAL_METH_FOHIDE;OPAL_METH_FOSHOW;OPAL_METH_FOBRTT;OPAL_TYPE_APPLICATION;',0,"Defaultoverview",)} **Related Topics**

isVisible example

In the following example, the **pushButton** method for the *siteToTop* button attempts to attach to an open form. If the attach is successful, the method checks to see if the form is visible. If the form is visible, the method makes it the top window:

```
; siteToTop::pushButton
method pushButton(var eventInfo Event)
var
  siteForm Form
endVar
; if form is on desktop
if siteForm.attach("Form : SITENOTE.FSL") then
  if siteForm.isVisible() then     ; if form is visible
    siteForm.bringToTop()          ; make it the topmost layer
  else
    msgStop("Sorry", "Can't see Sitenote form.")
  endif
endif
endMethod
```

keyChar method

Sends an event to a form's **keyChar** method.

Syntax

```
1. keyChar ( const aChar SmallInt, const vChar SmallInt, const state SmallInt ) Logical

2. keyChar ( const characters String [ , const state SmallInt ] ) Logical
```

Description

keyChar sends an event to a form's **keyChar** method. For Syntax 1, you must specify the ANSI character code in *aChar*, the virtual key code in *vChar*, and the keyboard state in *state* (using KeyboardStates constants). For Syntax 2, you can specify a string of one or more characters and, optionally, use the KeyBoardStates constants to specify a keyboard state.

Example

{button ,AL(`OPAL_TYPE_FORM;OPAL_METH_FOKEYPH;',0,"Defaultoverview",)} **Related Topics**

keyChar example

In the following example, a form named *Otherfrm* is already open and contains one field named *fieldOne*. The form-level **keyChar** method for *Otherfrm* echoes characters to *fieldOne*. The **pushButton** method of a button named *callOtherKeyC* on the current form attaches to *Otherfrm* as *otherForm,* calls the keyChar method for *otherForm*, and passes it a string.

The following is the code for the **pushButton** method for *callOtherKeyC* on the current form:

```
; callOtherKeyC::pushButton
method pushButton(var eventInfo Event)
var
  otherForm Form
endVar
; attach to the other form (assumes it's open)
if otherForm.attach("Form : OTHERFRM.FSL") then
  otherForm.keyChar("Hi! ")   ; send a string
else
  msgStop("Error", "The other form is not available.")
endif
endMethod
```

The following code is attached to *Otherfrm*'s form-level **keyChar** method:

```
; thisForm::keyChar  (OTHERFRM.FSL)
method keyChar(var eventInfo KeyEvent)
if eventInfo.isPreFilter()
  then
    ; code here executes for each object in form
  else
    ; code here executes just for form itself
    ; send the key on to fieldOne
    msgInfo("Status", "Executing Otherfrm's keychar.")
    fieldOne.keyChar(eventInfo.char())
endif
endMethod
```

keyPhysical method

Sends an event to a form's **keyPhysical** method.

Syntax

```
keyPhysical ( const aChar SmallInt, const vChar SmallInt, const state SmallInt ) Logical
```

Description

keyPhysical sends an event to a form's **keyPhysical** method. You must specify the ANSI character code in *aChar*, the virtual key code in *vChar*, and the keyboard state in *state* (using KeyboardStates constants).

Example

{button ,AL(`OPAL_TYPE_FORM;OPAL_METH_FOKEYCH;',0,"Defaultoverview",)} Related Topics

keyPhysical example

In the following example, a form named *OtherFr2* is already open, and it contains one field named *fieldOneThere*. The form-level **keyPhysical** method for *Otherfrm* echoes characters to *fieldOneThere*. The **keyPhysical** method of a field named *fieldOneHere* on the current form attaches to *Otherfrm* as *otherForm*. The method then calls the **keyPhysical** method for *otherForm*, and passes it the ANSI code of the character or keystroke, the virtual ANSI code of the character or keypress, and the keyboard state.

The following code is attached to the **keyPhysical** method for *fieldOneHere* on the current form:

```
; fieldOneHere::keyPhysical    (current form)
method keyPhysical(var eventInfo KeyEvent)
var
  otherForm Form
endVar
; attach to the other form (assumes it's open)
if otherForm.attach("Form : OTHERFR2.FSL") then
  ; switch statement sorts out keyBoardState
  switch
    case eventInfo.isShiftKeyDown() :
      otherForm.keyPhysical(eventInfo.charAnsiCode(),
                            eventInfo.vCharCode(), Shift)
    case eventInfo.isAltKeyDown() :
      otherForm.keyPhysical(eventInfo.charAnsiCode(),
                            eventInfo.vCharCode(),
                            Alt)
    case eventInfo.isControlKeyDown() :
      otherForm.keyPhysical(eventInfo.charAnsiCode(),
                            eventInfo.vCharCode(),
                            Control)
    otherwise:
      otherForm.keyPhysical(eventInfo.charAnsiCode(),
                            eventInfo.vCharCode(),
                            0)


  endSwitch
else
  msgStop("Error", "The other form is not available.")
endif
endMethod
```

The following code is attached to the **keyPhysical** method for *otherForm*:

```
; thisForm::keyPhysical   (OTHERFRM)
method keyPhysical(var eventInfo KeyEvent)
if eventInfo.isPreFilter()
  then
    ;code here executes for each object in form
  else
    ;code here executes just for form itself
    ; pass keyPhysical on to fieldOneThere
    ; switch statement sorts out keyBoardState
    switch
      case eventInfo.isShiftKeyDown() :
        fieldOneThere.keyPhysical(eventInfo.charAnsiCode(),
                                  eventInfo.vCharCode(), Shift)
      case eventInfo.isAltKeyDown() :
        fieldOneThere.keyPhysical(eventInfo.charAnsiCode(),
                                  eventInfo.vCharCode(), Alt)
      case eventInfo.isControlKeyDown() :
        fieldOneThere.keyPhysical(eventInfo.charAnsiCode(),
                                  eventInfo.vCharCode(), Control)
      otherwise :
        fieldOneThere.keyPhysical(eventInfo.charAnsiCode(),
                                  eventInfo.vCharCode(), 0)
    endSwitch
endif
endMethod
```

load method

Opens a form in the Form Design window.

Syntax

```
load ( const formName String, [const windowStyle LongInt [ , const x LongInt, const y LongInt,
const w LongInt, const h LongInt ] ] ) Logical
```

Description

load opens *formName* in the Form Design window. You have the option to specify in *windowStyle* a WindowStyles constant (or combination of constants). You also have the option to specify (in twips) the window's size and position: arguments *x* and *y* specify the position of the upper-left corner, arguments *w* and *h* specify the width and height, respectively. This method works only with saved forms (.FSL); it does not work with delivered forms (.FDL).

Compare this method to **open**, which opens a form in the Form window. To switch from the Form Design window to the Form window, use **run**. To switch from the Form window to the Form Design window, use **design**.

In either the Form Design window or the Form window, you can use UIObject type methods **create** and **methodSet** to place objects in the new form and attach methods to them. However, if you create objects while the form is in the Form window, the newly created objects will not automatically be saved when the form is closed.

Note

- Some form actions are especially processor-intensive. In some situations, you might need to follow a call to **open**, **load**, **design**, or **run** with a call to **sleep**. For more information, see the **sleep** procedure in the System type.

Example

{button ,AL(`OPAL_TYPE_FORM;OPAL_METH_FOCREATE;OPAL_METH_FOOPEN;OPAL_METH_FOOPAD;OPAL_METH_FODESIGN;',0,"Defaultoverview",)} Related Topics

load example

In the following example, the **pushButton** method for a button named *drawABox* loads the *Sitenote* form in a Form Design window. The method then sets the position of the form, creates a small box, names the box *newBox*, and sets its color to Blue. In the Form window, the box won't be visible; by default, the Visible property of objects created in this manner is False.

```
; drawABox::pushButton
method pushButton(var eventInfo Event)
var
   myForm Form
   newObj UIObject
endVar
; open Sitenote in a Form Design window
if myForm.load("Sitenote.fsl") then
   myForm.setPosition(720, 720, 1440*6, 1440*5)  ; 6" by 5"
   newObj.create(BoxTool, 1440, 1440*3, 360, 360, myForm)
   newObj.name = "newBox"
   newObj.color = Blue
else
   msgStop("Stop", "Couldn't load the form.")
endIf
endMethod
```

maximize method/procedure

Maximizes a window.

Syntax

```
maximize ( )
```

Description

maximize displays a window at its full size. Calling this method is equivalent to clicking Maximize on the Control menu.

Example

{button ,AL(`OPAL_TYPE_FORM;OPAL_METH_FOMINI;OPAL_METH_FOIMIN;OPAL_METH_FOIMAX;OPAL_METH_FOSHOW;',0,"Defaultoverview",)} Related Topics

maximize example

In the following example, the **pushButton** method for the *goSites* button opens the *Sitenote* form (assumed to be in the current database), minimizes the current form and then waits for a response. If *Sitenote* returns OK, this method maximizes the current form; otherwise, it restores the current form to its previous size.

```
; goSites::pushButton
method pushButton(var eventInfo Event)
var
  siteForm     Form
  returnString String
endVar
; open the Sitenote form, minimize self (this form) and then wait
siteForm.open("Sitenote")
minimize()
returnString = String(siteForm.wait())
; if siteForm returned "OK", then maximize--otherwise, restore
if returnString = "OK" then
  maximize()
  siteForm.close()
else
  show()
  siteForm.close()
endif
endMethod
```

The following code is attached to a button named *OKButton* on *Sitenote*:

```
; OKButton::pushButton
method pushButton(var eventInfo Event)
formReturn("OK")    ; return the string "OK" to the calling form
endMethod
```

menuAction method/procedure

Sends an event to a form's **menuAction** method.

Syntax

```
menuAction ( const action SmallInt ) Logical
```

Description

menuAction constructs a MenuEvent and calls a specified form's **menuAction** method. *action* is either one of the MenuCommand constants or a user-defined menu constant.

Note

- You can't use **menuAction** to send a MenuCommand constant that is equivalent to a File, New menu choice or a File, Open menu choice. To simulate these choices, call the appropriate ObjectPAL method (e.g., **create** {Form type} or **open** {TableView type}).

Example

{button ,AL(`OPAL_TYPE_FORM;OPAL_METH_FOACTI;',0,"Defaultoverview",)} Related Topics

menuAction example

In the following example, the *sendATile* button on the current form opens the *Sitenote* form and sends it a MenuWindowTile action.

```
; sendATile::pushButton
method pushButton(var eventInfo Event)
var
  siteForm  Form
endVar
if siteForm.open("Sitenote.fsl") then
  siteForm.menuAction(MenuWindowTile)
endif
endMethod
```

methodDelete method

Deletes a form-level method from a form.

Syntax

```
methodDelete ( const methodName String ) Logical
```

Description

methodDelete deletes a built-in or custom method specified in *methodName* from a form. You can also specify Var, Proc, Uses, or Const in *methodName* to clear the Var, Proc, Uses, or Const window of a form. If *methodName* is a built-in event method, the built-in behavior for that method is restored.

This method works only with saved forms (.FSL); it does not work with delivered forms (.FDL).

Example

{button ,AL(`OPAL_TYPE_FORM;OPAL_METH_FOMETHEDIT;OPAL_METH_FOMETHODGET;OPAL_METH_FOMETHODSET;OPAL_METH_UICREATE;OPAL_METH_UIMETHODGET;OPAL_METH_UIMETHODSET;',0,"Defaultoverview",)} Related Topics

methodDelete example

In the following example, two forms are on the desktop in a Form Design window: *Otherone* and *Othertwo*. The **pushButton** method for a button named *moveMethod* (on the current form) moves a method from *Otherone* to *Othertwo*.

```
; moveMethod::pushButton
method pushButton(var eventInfo Event)
var
  tempFormSrc,
  tempFormDest    Form
  transMethod String
endVar
; try to attach to both the source and the destination form
; assume source and destination are on the desktop in a Form Design window
if tempFormSrc.attach("Form Design : OTHERONE.FSL") AND
   tempFormDest.attach("Form Design : OTHERTWO.FSL") then
  ; get definition for source form's mouseRightUp, then delete
  transMethod = tempFormSrc.methodGet("mouseRightUp")
  tempFormSrc.methodDelete("mouseRightUp")
  ; copy the method to the destination form mouseRightUp
  tempFormDest.methodSet("mouseRightUp", transMethod)
else
  msgStop("Error", "Couldn't attach to source and destination forms.")
endif
endMethod
```

methodEdit method

Opens a form-level method in an Editor window.

Syntax

```
methodEdit (const methodName String) Logical
```

Description

methodEdit opens the method specified by *methodName* in an Editor window. If you try to open a method that doesn't exist, **methodEdit** will create it for you. **methodEdit** fails if you try to open a method that is running.

Example

{button ,AL(`OPAL_TYPE_FORM;OPAL_METH_FOMETHODGET;OPAL_METH_FOMETHODSET;OPAL_METH_FOMETHODDELETE;;',0,"Defaultoverview",)} Related Topics

methodEdit example

The following example opens the form's **testMethod** method in an Editor window:

```
method pushButton(var eventInfo Event)
var
    MyForm  form
endvar
MyForm.load("vendors.fsl")
MyForm.methodEdit("testMethod")
endMethod
```

methodGet method

Gets a form-level method.

Syntax

```
methodGet (const methodName String ) String
```

Description

methodGet gets the text of the built-in or custom form-level method specified in *methodName* attached to a form. You can also specify Var, Const, Uses, or Proc to get the contents of the Var, Const, Uses, or Proc window of a form.

This method works only with saved forms (.FSL); it does not work with delivered forms (.FDL).

Example

{button ,AL(`OPAL_TYPE_FORM;OPAL_METH_FOMETHEDIT;OPAL_METH_FOMETHODDELETE;OPAL_METH_FOMETHODSET;OPAL_METH_FOENUMSOURCETOFILE;OPAL_METH_UICREATE;OPAL_METH_UIMETHODGET;OPAL_METH_UIMETHODSET;',0,"Defaultoverview",)} **Related Topics**

methodGet example

See the **methodDelete** example.

methodSet method

Sets the definition of a method that is attached to a form.

Syntax

```
methodSet (const methodName String, const methodText String ) Logical
```

Description

methodSet writes the text in *methodText* to the built-in or custom form-level method *methodName* and overwrites any existing method definition. You can also specify Var, Const, Uses, or Proc to set the contents of the Var, Const, Uses, or Proc window of a form.

This method works only with saved forms (.FSL); it does not work with delivered forms (.FDL).

Example

{button ,AL(`OPAL_TYPE_FORM;OPAL_METH_FOMETHEDIT;OPAL_METH_FOMETHODDELETE;OPAL_METH_FOMETHODGET;OPAL_METH_UICREATE;OPAL_METH_UIMETHODGET;OPAL_METH_UIMETHODSET;',0,"Defaultoverview",)} Related Topics

methodSet example

See the **methodDelete** example.

minimize method/procedure

Minimizes a window.

Syntax

```
minimize ( )
```

Description

minimize displays a window as an icon. Calling this method is equivalent to choosing Minimize from the Control menu.

Example

{button ,AL(`OPAL_TYPE_FORM;OPAL_METH_FOMAXI;OPAL_METH_FOIMAX;OPAL_METH_FOIMIN;OPAL_METH_FOSHOW;OPAL_TYPE_APPLICATION;',0,"Defaultoverview",)} Related Topics

minimize example

See the **maximize** example.

mouseDouble method

Sends an event to a form's **mouseDouble** method.

Syntax

```
mouseDouble ( const x LongInt, const y LongInt, const state SmallInt ) Logical
```

Description

mouseDouble constructs a MouseEvent and sends it to a form's **mouseDouble** method. The arguments *x* and *y* specify (in twips) the location of the event, and *state* specifies a key state using KeyBoardStates constants.

Example

**{button ,AL(`OPAL_TYPE_FORM;OPAL_METH_FOMDOWN;OPAL_METH_FOMRDOU;',0,"Defaultoverview",)
} Related Topics**

mouseDouble example

In the following example, the form *Othermse* is open in the Form window. The **pushButton** method for a button named *sendMouseDouble* on the current form attaches to *Othermse* as *otherForm* and then calls the **mouseDouble** method for *otherForm*.

```
; sendMouseDouble::pushButton
method pushButton(var eventInfo Event)
var
  otherForm Form
endVar
; try to attach to target form
if otherForm.attach("Form : OTHERMSE.FSL") then
  ; send a mouseDouble to target form at coordinates 1000, 1000
  otherForm.mouseDouble(1000, 1000, LeftButton)
else
  msgStop("Quitting", "Could not find target form.")
endif
endMethod
```

The following code is attached to the **mouseDouble** method for *otherForm* (*Othermse*):

```
; otherMouse::mouseDouble  (OTHERMSE)
method mouseDouble(var eventInfo MouseEvent)
var
  targObj  UIObject
endVar
if eventInfo.isPreFilter()
  then
    ; code here executes for each object in form
  else
    ; code here executes just for form itself
    ; write method name to the lastMethod field
    lastMethod = "mouseDouble"
    ; get the target and write name to lastTarget field
    eventInfo.getTarget(targObj)
    lastTarget = targObj.Name
endif
endMethod
```

mouseDown method

Sends an event to a form's **mouseDown** method.

Syntax

```
mouseDown ( const x LongInt, const y LongInt, const state SmallInt ) Logical
```

Description

mouseDown constructs an event and sends it to a form's **mouseDown** method. The arguments *x* and *y* specify (in twips) the location of the event, and *state* specifies a key state using KeyBoardStates constants.

Example

{button ,AL(`OPAL_TYPE_FORM;OPAL_METH_FOMRDOW;OPAL_METH_FOMUP;',0,"Defaultoverview",)}

Related Topics

mouseDown example

In the following example, the form *Othermse* is open in the Form window. The **pushButton** method for a button named *sendMouseDown* on the current form attaches to *Othermse* as *otherForm* and then calls the **mouseDown** method for *otherForm*.

```
; sendMouseDown::pushButton
method pushButton(var eventInfo Event)
var
  otherForm Form
endVar
; try to attach to target form
if otherForm.attach("Form : OTHERMSE.FSL") then
  ; send a mouseDown to target form at coordinates 1000, 1000
  otherForm.mouseDown(1000, 1000, LeftButton)
else
  msgStop("Quitting", "Could not find target form.")
endif
endMethod
```

The following code is attached to the **mouseDown** method for *otherForm* (*Othermse*):

```
; otherMouse::mouseDown  (OTHERMSE)
method mouseDown(var eventInfo MouseEvent)
var
  targObj  UIObject
endVar
if eventInfo.isPreFilter()
  then
    ; code here executes for each object in form
  else
    ; code here executes just for form itself
    ; write method name to the lastMethod field
    lastMethod = "mouseDown"
    ; get the target and write name to lastTarget field
    eventInfo.getTarget(targObj)
    lastTarget = targObj.Name
endif
endMethod
```

mouseEnter method

Sends an event to a form's **mouseEnter** method.

Syntax

```
mouseEnter ( const x LongInt, const y LongInt, const state SmallInt ) Logical
```

Description

mouseEnter constructs a MouseEvent and sends it to a form's **mouseEnter** method. The arguments *x* and *y* specify (in twips) the location of the event, and *state* specifies a key state using KeyBoardStates constants.

Example

{button ,AL(`OPAL_TYPE_FORM;OPAL_METH_FOMEXIT;',0,"Defaultoverview",)} **Related Topics**

mouseEnter example

In the following example, the form *Othermse* is open in the Form window. The **pushButton** method for a button named *sendMouseEnter* on the current form attaches to *Othermse* as *otherForm* and then calls the **mouseEnter** method for *otherForm*.

```
; sendMouseEnter::pushButton
method pushButton(var eventInfo Event)
var
  otherForm Form
endVar
; try to attach to target form
if otherForm.attach("Form : OTHERMSE.FSL") then
  ; send a mouseEnter to target form at coordinates 1000, 1000
  otherForm.mouseEnter (1000, 1000, LeftButton)
else
  msgStop("Quitting", "Could not find target form.")
endif
endMethod
```

The following code is attached to the **mouseEnter** method for *otherForm* (*Othermse*):

```
; otherMouse::mouseEnter  (Othermse)
method mouseEnter(var eventInfo MouseEvent)
var
  targObj  UIObject
endVar
if eventInfo.isPreFilter()
  then
    ; code here executes for each object in form
  else
    ; code here executes just for form itself
    ; write method name to the lastMethod field
    lastMethod = "mouseEnter"
    ; get the target and write name to lastTarget field
    eventInfo.getTarget(targObj)
    lastTarget = targObj.Name
endif
endMethod
```

mouseExit method

Sends an event to a form's **mouseExit** method.

Syntax

```
mouseExit ( const x LongInt, const y LongInt, const state SmallInt ) Logical
```

Description

mouseExit constructs a MouseEvent and sends it to a form's **mouseExit** method. The arguments *x* and *y* specify (in twips) the location of the event, and *state* specifies a key state using KeyBoardStates constants.

Example

{button ,AL(`OPAL_TYPE_FORM;OPAL_METH_FOMENTER;',0,"Defaultoverview",)} Related Topics

mouseExit example

In the following example, the form *Othermse* is open in the Form window. The **pushButton** method for a button named *sendMouseExit* on the current form attaches to *Othermse* as *otherForm* and then calls the **mouseExit** method for *otherForm*.

```
; sendMouseExit::pushButton
method pushButton(var eventInfo Event)
var
  otherForm Form
endVar
; try to attach to target form
if otherForm.attach("Form : OTHERMSE.FSL") then
  ; send a mouseExit to target form at coordinates 1000, 1000
  otherForm.mouseExit(1000, 1000, LeftButton)
else
  msgStop("Quitting", "Could not find target form.")
endif
endMethod
```

The following code is attached to the **mouseExit** method for *otherForm* (*Othermse*):

```
; otherMouse::mouseExit  (Othermse)
method mouseExit(var eventInfo MouseEvent)
var
  targObj  UIObject
endVar
if eventInfo.isPreFilter()
  then
    ; code here executes for each object in form
  else
    ; code here executes just for form itself
    ; write method name to the lastMethod field
    lastMethod = "mouseExit"
    ; get the target and write name to lastTarget field
    eventInfo.getTarget(targObj)
    lastTarget = targObj.Name
endif
endMethod
```

mouseMove method

Sends an event to a form's **mouseMove** method.

Syntax

```
mouseMove ( const x LongInt, const y LongInt, const state SmallInt ) Logical
```

Description

mouseMove constructs an event and sends it to a form's **mouseMove** method. The arguments *x* and *y* specify (in twips) the location of the event, and *state* specifies a key state using KeyBoardStates constants.

Example

{button ,AL(`OPAL_TYPE_FORM;OPAL_METH_FOMENTER;OPAL_METH_FOMEXIT;',0,"Defaultoverview",)}

Related Topics

mouseMove example

In the following example, the form *Othermse* is open in the Form window. The **pushButton** method for a button named *sendMouseMove* on the current form attaches to *Othermse* as *otherForm* and then calls the **mouseMove** method for *otherForm*.

```
; sendMouseMove::pushButton
method pushButton(var eventInfo Event)
var
  otherForm Form
endVar
; try to attach to target form
if otherForm.attach("Form : OTHERMSE.FSL") then
  ; send a mouseMove to target form at coordinates 1000, 1000
  otherForm.mouseMove(1000, 1000, LeftButton)
else
  msgStop("Quitting", "Could not find target form.")
endif
endMethod
```

The following code is attached to the **mouseMove** method for *otherForm* (*Othermse*):

```
; otherMouse::mouseMove  (Othermse)
method mouseMove(var eventInfo MouseEvent)
var
  targObj  UIObject
endVar
if eventInfo.isPreFilter()
  then
    ; code here executes for each object in form
  else
    ; code here executes just for form itself
    ; write method name to the lastMethod field
    lastMethod = "mouseMove"
    ; get the target and write name to lastTarget field
    eventInfo.getTarget(targObj)
    lastTarget = targObj.Name
endif
endMethod
```

mouseRightDouble method

Sends an event to a form's **mouseRightDouble** method.

Syntax

```
mouseRightDouble (const x LongInt, const y LongInt, const state SmallInt ) Logical
```

Description

mouseRightDouble constructs a MouseEvent and sends it to a form's **mouseRightDouble** method. The arguments *x* and *y* specify (in twips) the location of the event, and *state* specifies a key state using KeyBoardStates constants.

Example

**{button ,AL(`OPAL_TYPE_FORM;OPAL_METH_FOMRDOU;OPAL_METH_FOMDOWN;',0,"Defaultoverview",)
} Related Topics**

mouseRightDouble example

In the following example, the form *Othermse* is open in the Form window. The **pushButton** method for a button named send *MouseRightDouble* on the current form attaches to *Othermse* as *otherForm* and then calls the **mouseRightDouble** method for *otherForm*.

```
; mouseRightDouble::pushButton
method pushButton(var eventInfo Event)
var
  otherForm Form
endVar
; try to attach to target form
if otherForm.attach("Form : OTHERMSE.FSL") then
  ; send a mouseRightDouble to target form at coordinates 1000, 1000
  otherForm.mouseRightDouble(1000, 1000, RightButton)
else
  msgStop("Quitting", "Could not find target form.")
endif
endMethod
```

The following code is attached to the **mouseRightDouble** method for *otherForm* (*Othermse*):

```
; otherMouse::mouseRightDouble  (Othermse)
method mouseRightDouble(var eventInfo MouseEvent)
var
  targObj  UIObject
endVar
if eventInfo.isPreFilter()
  then
    ; code here executes for each object in form
  else
    ; code here executes just for form itself
    ; write method name to the lastMethod field
    lastMethod = "mouseRightDouble"
    ; get the target and write name to lastTarget field
    eventInfo.getTarget(targObj)
    lastTarget = targObj.Name
endif
endMethod
```

mouseRightDown method

Sends an event to a form's **mouseRightDown** method.

Syntax

```
mouseRightDown ( const x LongInt, const y LongInt, const state SmallInt ) Logical
```

Description

mouseRightDown constructs a MouseEvent and sends it to a form's **mouseRightDown** method. The arguments *x* and *y* specify (in twips) the location of the event, and *state* specifies a key state using KeyBoardStates constants.

Example

**{button ,AL(`OPAL_TYPE_FORM;OPAL_METH_FOMRDOU;OPAL_METH_FOMDOWN;',0,"Defaultoverview",)
} Related Topics**

mouseRightDown example

In the following example, the form *Othermse* is open in the Form window. The **pushButton** method for a button named *sendMouseRightDown* on the current form attaches to *Othermse* as *otherForm* and then calls the **mouseRightDown** method for *otherForm*.

```
; mouseRightDown::pushButton
method pushButton(var eventInfo Event)
var
  otherForm Form
endVar
; try to attach to target form
if otherForm.attach("Form : OTHERMSE.FSL") then
  ; send a mouseRightDown to target form at coordinates 1000, 1000
  otherForm.mouseRightDown(1000, 1000, RightButton)
else
  msgStop("Quitting", "Could not find target form.")
endif
endMethod
```

The following code is attached to the **mouseRightDown** method for *otherForm* (*Othermse*):

```
; otherMouse::mouseRightDown  (Othermse)
method mouseRightDown(var eventInfo MouseEvent)
var
  targObj  UIObject
endVar
if eventInfo.isPreFilter()
  then
    ; code here executes for each object in form
  else
    ; code here executes just for form itself
    ; write method name to the lastMethod field
    lastMethod = "mouseRightDown"
    ; get the target and write name to lastTarget field
    eventInfo.getTarget(targObj)
    lastTarget = targObj.Name
endif
endMethod
```

mouseRightUp method

Sends an event to a form's **mouseRightUp** method.

Syntax

```
mouseRightUp ( const x LongInt, const y LongInt, const state SmallInt ) Logical
```

Description

mouseRightUp constructs a MouseEvent and sends it to a form's **mouseRightUp** method. The arguments *x* and *y* specify (in twips) the location of the event, and *state* specifies a key state using KeyBoardStates constants.

Example

{button ,AL(`OPAL_TYPE_FORM;OPAL_METH_FOMUP;OPAL_METH_FOMRDOW;',0,"Defaultoverview",)}

Related Topics

mouseRightUp example

In the following example, assume the form *Othermse* is already open. The **pushButton** method for a button named *sendMouseRightUp* on the current form attaches to *Othermse* as *otherForm* and then calls the **mouseRightUp** method for *otherForm*.

```
; mouseRightUp::pushButton
method pushButton(var eventInfo Event)
var
  otherForm Form
endVar
; try to attach to target form
if otherForm.attach("Form : OTHERMSE.FSL") then
  ; send a mouseRightUp to target form at coordinates 1000, 1000
  otherForm.mouseRightUp(1000, 1000, RightButton)
else
  msgStop("Quitting", "Could not find target form.")
endif
endMethod
```

The following code is attached to the **mouseRightUp** method for *otherForm* (*Othermse*):

```
; otherMouse::mouseRightUp  (Othermse)
method mouseRightUp(var eventInfo MouseEvent)
var
  targObj  UIObject
endVar
if eventInfo.isPreFilter()
  then
    ; code here executes for each object in form
  else
    ; code here executes just for form itself
    ; write method name to the lastMethod field
    lastMethod = "mouseRightUp"
    ; get the target and write name to lastTarget field
    eventInfo.getTarget(targObj)
    lastTarget = targObj.Name
endif
endMethod
```

mouseUp method

Sends an event to a form's **mouseUp** method.

Syntax

```
mouseUp ( const x LongInt, const y LongInt, const state SmallInt ) Logical
```

Description

mouseUp constructs a MouseEvent and sends it to a form's **mouseUp** method. The arguments *x* and *y* specify (in twips) the location of the event, and *state* specifies a key state using KeyBoardStates constants.

Example

{button ,AL(`OPAL_TYPE_FORM;OPAL_METH_FOMRUP;OPAL_METH_FOMDOWN;',0,"Defaultoverview",)}

Related Topics

mouseUp example

In the following example, the form *Othermse* is open in the Form window. The **pushButton** method for a button named *sendMouseUp* on the current form attaches to *Othermse* as *otherForm* and then calls the **mouseUp** method for *otherForm*.

```
; sendMouseUp::pushButton
method pushButton(var eventInfo Event)
var
  otherForm Form
endVar
; try to attach to target form
if otherForm.attach("Form : OTHERMSE.FSL") then
  ; send a mouseUp to target form at coordinates 1000, 1000
  otherForm.mouseUp(1000, 1000, LeftButton)
else
  msgStop("Quitting", "Could not find target form.")
endif
endMethod
```

The following code is attached to the **mouseUp** method for *otherForm* (*Othermse*):

```
; otherMouse::mouseUp  (Othermse)
method mouseUp(var eventInfo MouseEvent)
var
  targObj  UIObject
endVar
if eventInfo.isPreFilter()
  then
    ; code here executes for each object in form
  else
    ; code here executes just for form itself
    ; write method name to the lastMethod field
    lastMethod = "mouseUp"
    ; get the target and write name to lastTarget field
    eventInfo.getTarget(targObj)
    lastTarget = targObj.Name
endif
endMethod
```

moveTo method

Moves to a form.

Syntax

```
moveTo ( [const objectName String] ) Logical
```

Description

moveTo moves the focus to a form. Optionally, it moves to the object specified in *objectName*.

Example

{button ,AL(`OPAL_TYPE_FORM;OPAL_METH_FOMOTP;',0,"Defaultoverview",)} **Related Topics**

moveTo example

In the following example, a form named *Sitenote* is already open in the Form window. The **pushButton** method for the *goToSites* button in the current form attaches the variable *otherForm* to *Sitenote*, determines if *otherForm* is visible, and, if so, moves to *otherForm*. If *otherForm* is not visible, the method uses **show** to display the form at its default size (**show** also moves the focus to the target form).

```
; goToSites::pushButton
method pushButton(var eventInfo Event)
var
  otherForm  Form
endVar
; assume that Sitenote form is already open
if otherForm.attach("Form : SITENOTE.FSL") then
  if otherForm.isVisible() then
    otherForm.moveTo()       ; if form is visible, move to it
  else
    otherForm.show()         ; otherwise, make it visible
  endif
else
  msgStop("Stop", "Couldn't find form.")
endif
endMethod
```

moveToPage method/procedure

Displays a specified page of a form.

Syntax

```
moveToPage ( const pageNumber SmallInt ) Logical
```

Description

moveToPage displays the page of a form specified in *pageNumber*. *pageNumber* can be an integer variable or an integer constant, but it can't be an object ID. To move to a page by using its object ID, use the **moveTo** method from the UIObject type.

Example

{button ,AL(`OPAL_TYPE_FORM;OPAL_METH_FOBRTT;',0,"Defaultoverview",)} **Related Topics**

moveToPage example

In the following example, the current form has two pages. The *Sitenote* form exists in the working directory and has four pages. The **pushButton** method for *pageThruSites* (on the current form) moves to the second page of the current form, opens the *Sitenote* form to the *otherForm* variable, and pages through *otherForm*.

```
; pageThruSites::pushButton
method pushButton(var eventInfo Event)
const
   BillingInfo = SmallInt(4)
endConst
var
   myForm, otherForm  Form
   somePage           SmallInt
endVar
moveToPage(2)                       ; moves to page 2 on this form
if otherForm.open("Sitenote.fsl") then  ; opens to first page
  sleep(2000)                       ; pause
  otherForm.moveToPage(2)           ; moves to page 2 of SiteNote
  sleep(2000)
  somePage = 3
  otherForm.moveToPage(somePage)    ; moves to page 3
  sleep(2000)
  otherForm.moveToPage(BillingInfo) ; moves to page 4
  sleep(2000)
endIf
endMethod
```

open method

Opens a window.

Syntax

```
1. open ( const formName String [ , const windowStyle LongInt ] ) Logical

2. open ( const formName String, const windowStyle LongInt, const x SmallInt, const y SmallInt,
const w SmallInt, const h SmallInt ) Logical

3. open ( const openInfo FormOpenInfo ) Logical
```

Description

open displays the form specified in *formName*. The form is opened in a Form window. The optional arguments *x* and *y* specify the location of the upper-left corner of the form (in twips), *w* and *h* specify the width and height (in twips), and *windowStyle* specifies display attributes using WindowStyles constants. You can specify more than one window style element by adding the constants together. The following code opens a form and specifies both vertical and horizontal scroll bars:

```
theForm.open("sales", WinStyleDefault + WinStyleVScroll + WinStyleHScroll)
```

Compare this method with **load**, which opens a form in a Form Design window.

Syntax 3 lets you specify form settings from *openInfo*, a record of type FormOpenInfo. The predefined FormOpenInfo record has the following structure:

```
x, y, w, h     LongInt ;position and size of the form
name           String  ;name of form to open
masterTable    String  ;new master table name
queryString    String  ;query to run (actual query string)
SQLString      String  ;SQL query to run (actual query string)
windowStyle    LongInt ;window style constant(s)
```

You can use the *masterTable* member to specify a different master table for the form (this is similar to choosing a different table for a form when you open the form from the Open Form dialog box). Alternatively, you can specify a query string in the *queryString* member. If the query string is an SQL query, replace *queryString* with *SQLString*. Corel Paradox executes the query and opens the form; the result of the query is the master table.

Corel Paradox opens saved forms before delivered forms with the same name. For example, suppose the working directory contains ORDERS.FSL (a saved form) and ORDERS.FDL (a delivered form). The following statement opens the saved form, ORDERS.FSL.

```
ordersForm.open("ORDERS") ; Opens :WORK:ORDERS.FSL.
```

To specify a delivered form, include the .FDL extension.

```
ordersForm.open("ORDERS.FDL") ; Opens the delivered form.
```

In addition to being a table name for a QBE file, the MasterTable field may be the name of a SQL file that produces an Answer table.

FormOpenInfo now has a new field, SQLString, which can be used to specify an SQL statement to execute. SQLString is of type String.

To rebind a report to a newly created SQL statement, save the SQL statement to a file and specify the filename in ReportPrintInfo.MasterTable or ReportOpenInfo.MasterTable.

Note

- Some form actions are especially processor-intensive. In some situations, you might need to follow a call to **open**, **load**, **design**, or **run** with a **sleep**. For more information, see the **sleep** procedure in the System type.

Examples

{button ,AL(`OPAL_TYPE_FORM;OPAL_METH_FOCLOS;OPAL_METH_REPRINT;OPAL_METH_FOCREATE;OPAL_METH_FOLOAD;OPAL_METH_FOOPAD;OPAL_METH_FODESIGN;',0,"Defaultoverview",)} Related Topics

open method examples

Example1 Using **keyPhysical** to open a form

Example2 Using **FormOpenInfo** to set the characteristics of the form opened

open example 1

In the following example, the **keyPhysical** method for a field named *fieldOne* tests all key events. When the user presses F1, the form HELPFORM opens. The **keyPhysical** method opens a form from the current directory:

```
; fieldOne::keyPhysical
method keyPhysical(var eventInfo KeyEvent)
var
  helpForm Form
endVar
message(eventInfo.vChar())
if eventInfo.vChar() = "VK_F1" then
  helpForm.open("helpform", WinStyleDefault,
                720, 720, 1440 * 2, 1440 * 4)
  disableDefault
endIf


endMethod
```

open example 2

The following example works like the previous example, except that it uses a FormOpenInfo record to set the characteristics of the form to be opened.

```
; fieldOne::keyPhysical
method keyPhysical(var eventInfo KeyEvent)
var
  openHelpForm FormOpenInfo   ; a predeclared record type
  helpForm     Form
endVar
message(eventInfo.vChar())
if eventInfo.vChar() = "VK_F1" then
  openHelpForm.x = 720
  openHelpForm.y = 720
  openHelpForm.w = 2 * 1440
  openHelpForm.h = 4 * 1440
  openHelpForm.name = "helpform"
  helpForm.open(openHelpForm)
  disableDefault
endIf
endMethod
```

openAsDialog method

Opens a Form window as a dialog box.

Syntax

```
1. openAsDialog ( const formName [ , const windowStyle LongInt] ) Logical

2. openAsDialog ( const formName String, const windowStyle LongInt, const x SmallInt, const y
SmallInt, const w SmallInt, const h SmallInt ) Logical

3. openAsDialog ( const openInfo FormOpenInfo ) Logical
```

Description

openAsDialog opens the form *formName* and displays it on top of any other open windows. The form is in the Form window. *formName* is always on top, whether it's active or not. The optional arguments *x* and *y* specify the upper-left corner of the window (in twips), *w* and *h* specify the width and height (in twips), and *windowStyle* specifies display attributes using WindowStyles constants. You can specify more than one window style element by adding the constants. The following code opens a form and specifies both vertical and horizontal scroll bars:

```
theForm.openAsDialog("sales", WinStyleDefault + WinStyleVScroll + WinStyleHScroll)
```

Syntax 3 lets you specify form settings from *openInfo*, a record of type FormOpenInfo. The FormOpenInfo record type is predeclared and has the following structure:

```
x, y, w, h      LongInt  ; position and size of the form
name            String   ; name of form to open
masterTable     String   ; master table name
queryString     String   ; run this query
```

Example

{button ,AL(`OPAL_TYPE_FORM;OPAL_METH_FOOPEN;OPAL_METH_FOWAIT;OPAL_METH_FOFORMCALLER ;OPAL_METH_FOFORR;',0,"Defaultoverview",)} Related Topics

openAsDialog example

In the following example, the **keyPhysical** method for a field named *fieldOne* tests all key events. When the user presses F1, the form HELPFORM opens. The **keyPhysical** method opens a form as a dialog box.

```
; fieldOne::keyPhysical
method keyPhysical(var eventInfo KeyEvent)
var
  helpForm Form
endVar
; if user presses F1, open a help dialog box
if eventInfo.vChar() = "VK_F1" then


  helpForm.openAsDialog("helpform", WinStyleDefault,
                        720, 720, 1440 * 4, 1440 * 3)


  helpForm.setTitle("Application Help")
  helpForm.wait()
  helpForm.close()
  disableDefault                        ; don't call Help system
endIf
endMethod
```

postAction method

Posts an action to an action queue for delayed execution.

Syntax

```
postAction ( const actionId SmallInt )
```

Description

postAction works like **action**, except that the action is not executed immediately. Instead, the action specified by *actionID* is posted to an action queue at the time of the method call; Corel Paradox waits until a yield occurs (e.g., by the current method completing execution or by a call to **sleep**).

The value of *actionID* can be a user-defined action constant or a constant from one of the following Action classes:

ActionDataCommands
ActionEditCommands
ActionFieldCommands
ActionMoveCommands
ActionSelectCommands

Example

{button ,AL(`OPAL_TYPE_FORM;OPAL_METH_FOACTI;',0,"Defaultoverview",)} **Related Topics**

postAction example

In the following example, the **pushButton** method for *openSitesNew* opens the *Sitenote* form to the variable *otherForm*. The method then posts three actions to *otherForm* and displays a message in a dialog box. The actions specified by **postAction** occur when Corel Paradox yields:

```
; openSitesNew::pushButton
method pushButton(var eventInfo Event)
; otherForm variable is global to form--stays in scope after method ends
if otherForm.open("Sitenote.fsl") then
  ; these actions will not execute until after this method ends
  otherForm.postAction(DataEnd)          ; move to the last record
  otherForm.postAction(DataBeginEdit)    ; start Edit mode
  otherForm.postAction(DataInsertRecord) ; insert a new blank record
  msgInfo("Status", "About to perform posted actions. Watch closely.")
else
  msgStop("Stopped", "Could not open form.")
endif
endMethod
```

run method

Switches a form from the Form Design window to the Form window.

Syntax

run ( ) Logical

Description

run switches a form from the Form Design window to the Form window. This method works only with saved forms (.FSL); it does not work with delivered forms (.FDL).

To switch from the Form window to the Form Design window, use **design**.

Note

- Some form actions are especially processor-intensive. In some situations, you might need to follow a call to **open**, **load**, **design**, or **run** with a call to **sleep**. For more information, see the **sleep** method in the System type.

Example

{button ,AL(`OPAL_TYPE_FORM;OPAL_METH_FODESIGN;',0,"Defaultoverview",)} Related Topics

run example

The following example opens the *Sitenote* form in a Form Design window, deletes the **pushButton** method from the form and then runs the form. Assume that the *Sitenote* form is in the current directory. This code is attached to the **pushButton** code for *delPushButton*.

```
; delPushButton::pushButton
method pushButton(var eventInfo Event)
var
  otherForm  Form
endVar
; load the Sitenote form, delete the pushButton
; method, then run the form
if otherForm.load("Sitenote") then
  otherForm.methodDelete("pushButton")
  otherForm.run()
endif               ; won't be permanent
endMethod
```

save method

Saves a form to disk.

Syntax

save ([const ***newFormName*** String] **)** Logical

Description

save writes a form to disk in the user's working directory. This method works only when the form is in a Form Design window.

The *newFormName* argument specifies the name for the form. If the form already has a name, Corel Paradox saves the form using that name. If you omit *newFormName* and the form doesn't have a name already, this method returns an error.

Example

{button ,AL(`OPAL_TYPE_FORM;OPAL_METH_FOCREATE;OPAL_METH_FODESIGN;',0,"Defaultoverview",)}

Related Topics

save example

See the **create** example.

saveStyleSheet method

Saves a style sheet.

Syntax

saveStyleSheet (const ***fileName*** String, const ***overWrite*** Logical **)** Logical

Description

saveStyleSheet saves a style sheet to the file specified in *fileName*. If *fileName* does not specify a full path for the style sheet file, this method saves it to the working directory.

The value of *overWrite* specifies what to do if the file already exists. If *overWrite* is True and the file exists, Corel Paradox overwrites the file without asking for confirmation. If *overWrite* is False and the file exists, the file is not saved. This method returns True if it saves the file; otherwise, it returns False.

saveStyleSheet saves the form's current style sheet, including any changes made interactively, or by using ObjectPAL. If called as a method, **saveStyleSheet** operates on the specified form. If called as a procedure, **saveStyleSheet** operates on the current form. It returns True if successful; otherwise, it returns False.

Example

{button ,AL(`OPAL_TYPE_FORM;OPAL_METH_FOGETSTYLESHEET;OPAL_METH_FOSETSTYLESHEET;OPAL_METH_FOGETPROTOPROPERTY;OPAL_METH_FOSETPROTOPROPERTY;',0,"Defaultoverview",)} Related Topics

saveStyleSheet example

The following example sets the frame style of field objects and text objects and then saves the form's style sheet to a file named IN3DFRAM.FT. If the file exists, it is overwritten.

```
const
   kOverWrite = Yes
endConst


method mouseClick(var eventInfo MouseEvent)
   var
      thisForm Form
   endVar


   thisForm.attach()
   thisForm.setProtoProperty(FieldTool, "Frame.Style", Inside3DFrame)
   thisForm.setProtoProperty(TextTool, "Frame.Style", Inside3DFrame)
   thisForm.saveStyleSheet("in3dfram.ft", kOverWrite)
endmethod
```

selectCurrentTool method

Specifies a Toolbar tool to use.

Syntax

```
selectCurrentTool ( const objType SmallInt ) Logical
```

Description

selectCurrentTool specifies which Toolbar tool to use, where *objType* is one of the UIObjectTypes constants. When used with a form in the Form Design window, this method makes the specified tool active and sets the mouse shape accordingly.

Example

{button ,AL(`OPAL_TYPE_FORM;OPAL_METH_FOSETSELECTEDOBJECTS;',0,"Defaultoverview",)} Related Topics

selectCurrentTool example

The following example creates a form and sets the current tool to the Field tool.

```
method pushButton(var eventInfo Event)
   var
      foTest   Form
   endVar


   foTest.create()
   foTest.selectCurrentTool(FieldTool)
   msgInfo("Next step:",
           "Click and drag to draw a field object.")
endMethod
```

setCompileWithDebug method

Sets Compile With Debug.

Syntax

```
setCompileWithDebug ( const yesNo Logical ) Logical
```

Description

setCompileWithDebug sets the Compile With Debug flag to true or false. This is the same as setting Compile With Debug interactively in a Form Design window. **setCompileWithDebug** returns True if successful; otherwise, it returns False.

Example

{button ,AL(`OPAL_TYPE_FORM;OPAL_METH_FOISCOMPILEWITHDEBUG;OPAL_METH_SYDEBUG;',0,"Defaultoverview",)} Related Topics

setCompileWithDebug example

In the following example, the central form of a management system has two buttons: *getCompileStatus* and *setCompileStatus*. The **pushButton** method of each button opens the Windows 95 Explorer dialog box to allow a user to select the file that will be examined or manipulated. Each method analyzes the fileName selected to determine the fileType and opens the file under the appropriate object type.

The following code is attached to the **pushButton** method for *setCompileStatus*:

```
; setCompileStatus::pushButton
method pushButton(var eventInfo Event)
var
   theForm       Form             ;// Object variable for forms
   theLibrary    Library          ;// Object variable for libraries
   theScript     Script           ;// Object variable for scripts
   fbi           FileBrowserInfo  ;// File Browser information structure
   selectedFile  String           ;// FileName selected by user
   fileType      String           ;// File type of file
                                  ;// selected by user
   status        Logical          ;// Debug status of the selected file
   toggle        String           ;// User choice for
endVar


      ;//Set allowable file types: Forms, Libraries, and Scripts
   fbi.AllowableTypes = fbForm + fbLibrary + fbScript
   if fileBrowser(selectedFile, fbi) then
      ;// The user selected a file
      fileType = upper(substr(selectedFile, selectedFile.size() - 2, 3))
      switch
         case fileType = "FSL" :
            ;// Load the Form
            theform.load(fbi.Drive + fbi.Path + selectedFile)
            ;// Determine its status
            status = theForm.isCompileWithDebug()
            toggle = msgYesNoCancel ("Select a choice", selectedFile
                                     + iif(status, " is ", " is not ") +
                        "compiled with Debug information - toggle?")
            switch
               case toggle = "Yes" :
                  ;// Toggle status
                  theForm.setCompileWithDebug(NOT(status))
                  ;// Save the change
                  theForm.save()
                  msgInfo("User Notification",
                           "Toggle of Debug State Completed.")
               case toggle = "No" or toggle = "Cancel" :
                  msgInfo("User Notification",
                           "Toggle of Debug State Canceled.")
            endSwitch
            ;// Close the Form
            theForm.close()


         case fileType = "LSL" :
            ;// Load the Library
            theLibrary.load(fbi.Drive + fbi.Path + selectedFile)
            ;// Determine its status
            status = theLibrary.isCompileWithDebug()
            toggle = msgYesNoCancel ("Select a choice", selectedFile
                                     + iif(status, " is ", " is not ")
                        + "compiled with Debug information - toggle?")
            switch
               case toggle = "Yes" :
                  ;// Toggle status
                  theLibrary.setCompileWithDebug(NOT(status))
                  ;// Save the change
                  theLibrary.save()
                  msgInfo("User Notification",
                           "Toggle of Debug State Completed.")
               case toggle = "No" or toggle = "Cancel" :
                  msgInfo("User Notification",
```

```
                        "Toggle of Debug State Canceled.")
            endSwitch
            ;// Close the Library
            theLibrary.close()

         case fileType = "SSL" :
            ;// Load the Script
            theScript.load(fbi.Drive + fbi.Path + selectedFile)
            ;// Determine its status
            status = theScript.isCompileWithDebug()
            toggle = msgYesNoCancel ("Select a choice", selectedFile
                                + iif(status, " is ", " is not ")
                           + "compiled with Debug information - toggle?")
            switch
               case toggle = "Yes" :
                  ;// Toggle status
                  theScript.setCompileWithDebug(NOT(status))
                  ;// Save the change
                  theScript.save()
                  msgInfo("User Notification",
                           "Toggle of Debug State Completed.")
               case toggle = "No" or toggle = "Cancel" :
                  msgInfo("User Notification",
                           "Toggle of Debug State Canceled.")
            endSwitch
            ;// Close the Script
            theScript.close()
      endSwitch
      ;// Inform the user
      msgInfo(selectedFile
            + " compiled with Debug information?",
                 status)
   else
      ;// The user didn't select a file
      msgInfo("No file selected", "Please try again")
   endIf
endMethod
```

setIcon method/procedure

Specifies the icon to be used with a form, report, or desktop.

Syntax

```
setIcon ( const fileName String ) Logical
```

Description

setIcon specifies the icon to be used with a form, report, or desktop. The file specified with *fileName* must be a valid icon file and the file's name must have an extension of .ICO. **setIcon** returns True if successful; otherwise it returns False.

After you set the icon for a form, all the forms on the desktop will change to the new icon and any form that is opened will be set to the new icon.

Example

{button ,AL(`OPAL_TYPE_FORM;OPAL_METH_FOATTA;OPAL_METH_FOOPEN;OPAL_TYPE_APPLICATION;',0, "Defaultoverview",)} **Related Topics**

setIcon example

The following example sets the file, DOCFILE.ICO as the icon.

```
method init ( var eventInfo Event )
   setIcon ( "i:\\resource\\docfile.ico" )
endMethod
```

setMenu method

Associates a menu with a form.

Syntax

```
setMenu ( const menuVar Menu )
```

Description

setMenu associates the menu specified in *menuVar* with a form. This method performs the same function as the Menu type **show**, and adds the following features:

- when the form gets focus, Corel Paradox displays the associated menu
- actions that result from choices from that menu are sent to that form

Example

setMenu example

The following example is a script. It opens a form, builds a simple menu and then uses **setMenu** to assign the menu to the form:

```
method run(var eventInfo Event)
   var
      foOrders   Form
      muOrderForm   Menu
      puFormFile   PopUpMenu
   endVar


; Build a menu for the form.
   foOrders.open("orders")


; Setting the StandardMenu property to False
; (either in ObjectPAL code or interactively)
; can reduce flicker when changing menus.
   foOrders.StandardMenu = False


   puFormFile.addText("&New Form", MenuEnabled, MenuFormNew)
   puFormFile.addText("&Open Form", MenuEnabled, MenuFormOpen)
   puFormFile.addText("&Exit", MenuEnabled, MenuFileExit)


   muOrderForm.addPopUp("&File", puFormFile)


   foOrders.setMenu(muOrderForm)


endMethod
```

setPosition method/procedure

Positions a window on screen.

Syntax

```
setPosition ( const x LongInt, const y LongInt, const w LongInt, const h LongInt )
```

Description

setPosition positions a window on screen. The arguments *x* and *y* specify the coordinates of the upper-left corner of the form (in twips), and *w* and *h* specify the width and height (in twips).

To ObjectPAL, the screen is a two-dimensional grid, with the origin (0, 0) at the upper-left corner of an object's container, positive x-values extending to the right, and positive y-values extending down.

For dialog boxes and for the Corel Paradox desktop application, the position is given relative to the entire screen; for forms, reports, and Table windows, the position is given relative to the Corel Paradox desktop.

Example

**{button ,AL(`OPAL_TYPE_FORM;OPAL_METH_FOOPEN;OPAL_TYPE_APPLICATION;;;',0,"Defaultoverview",)
}** **Related Topics**

setPosition example

See the **getPosition** example.

setProtoProperty method/procedure

Sets the value of a specified property of a prototype object.

Syntax

```
setProtoProperty ( const objectType SmallInt, propertyName String, value AnyType ) Logical
```

Description

setProtoProperty sets the property specified in *propertyName* of the prototype object specified in *objectType* to the value specified in *value*. To specify *objectType*, use one of the UIObjectTypes constants. If called as a method, **setProtoProperty** operates on prototype objects in the style sheet of the specified form. If called as a procedure, **setProtoProperty** uses the style sheet of the current form.

Changes to the style sheet are not saved automatically. You must either save the style sheet interactively or call **saveStyleSheet**.

Example

{button ,AL(`OPAL_TYPE_FORM;OPAL_METH_FOGETPROTOPROPERTY;OPAL_METH_FOGETSTYLESHEET;OPAL_METH_FOSAVESTYLESHEET;',0,"Defaultoverview",)} Related Topics

setProtoProperty example

See the **saveStyleSheet** example.

setSelectedObjects method

Selects specified objects in a form.

Syntax

```
setSelectedObjects ( var objects Array[ ] UIObject, const yesNo Logical )
```

Description

setSelectedObjects selects specified objects in a form in a Form Design window as if you had selected the objects interactively. The array *objects* is an array of available UIObjects (not the object names). Use **attach** to assign a UIObject to an array.

The argument *yesNo* specifies whether to show selection handles If *yesNo* is True, the selected objects have handles; otherwise, they do not.

Example

{button ,AL(`OPAL_TYPE_FORM;OPAL_METH_FOGETSELECTEDOBJECTS;OPAL_METH_UIATTA;',0,"Defaultoverview",)} **Related Topics**

setSelectedObjects example

The following example creates a form, creates two boxes in it, and calls **setSelectedObjects** to select the boxes. You must use **attach** to assign a UIObject to an array.

```
method pushButton(var eventInfo Event)
   var
      foTemp      Form
      uiTemp      UIObject
      arObjects   Array[2] UIObject
   endVar


   const
      kOneInch = 1440 ; One inch = 1,440 twips.
      kShowHandles = Yes
   endConst


   foTemp.create()


   uiTemp.create(BoxTool, 300, 300, kOneInch, kOneInch, foTemp)
   uiTemp.Visible = Yes
   arObjects[1].attach(uiTemp)


   uiTemp.create(BoxTool, 300, 2200, kOneInch, kOneInch, foTemp)
   uiTemp.Visible = Yes
   arObjects[2].attach(uiTemp)


   foTemp.setSelectedObjects(arObjects, kShowHandles)
endMethod
```

setStyleSheet method/procedure

Specifies a form's style sheet.

Syntax

setStyleSheet (const ***fileName*** String **)**

Description

setStyleSheet makes a form use the style sheet specified in *fileName*. If *fileName* does not specify a full path to the style sheet, this method searches for it in the working directory. If called as a method, **setStyleSheet** operates on the specified form. If called as a procedure, it operates on the current form.

Any UIObjects created in the form while the style sheet is active will have the properties and methods of the corresponding prototype objects in the style sheet. **setStyleSheet** does not change the properties or methods of UIObjects that already exist. This method affects only the specified form; it does not affect the screen or printer style sheets. Use the System procedures **setDefaultScreenStyleSheet** and **setDefaultPrinterStyleSheet** to set the properties if the screen and printer style sheets.

Example

{button ,AL(`OPAL_TYPE_FORM;OPAL_METH_FOGETSTYLESHEET;OPAL_METH_FOSAVESTYLESHEET;OPAL_METH_FOGETPROTOPROPERTY;OPAL_METH_FOSETPROTOPROPERTY;OPAL_METH_SYSETDEFAULTPRINTERSTYLESHEET;OPAL_METH_SYSETDEFAULTSCREENSTYLESHEET;',0,"Defaultoverview",)} Related Topics

setStyleSheet example

The following example opens a form and then calls **getStyleSheet** to see which style sheet the form is using. If the style sheet is not COREL.FT, the code calls **setStyleSheet** to set it and then calls **getStyleSheet** again to make sure it was set successfully. **setStyleSheet** requires double backslashes in the path, but **getStyleSheet** returns single backslashes.

```
method pushButton(var eventInfo Event)
   var
      f Form
   endVar
   f.open("orders")
   ; Get and set the style sheet for this form.
   if f.getStyleSheet() <> "c:\\Corel\\Suite8\\Paradox\\Corel.ft" then
      f.setStyleSheet("c:\\Corel\\Suite8\\Paradox\\Corel.ft")
      if f.getStyleSheet() <> "c:\\Corel\\Suite8\\Paradox\\Corel.ft" then
         msgStop("Problem", "Could not set the style sheet.")
       endIf
   endIf
endMethod
```

setTitle method/procedure

Sets the text in the Title Bar of the window.

Syntax

```
setTitle ( const text String )
```

Description

setTitle changes the text of the window's Title Bar to the text specified in *text*. The maximum length of *text* is 78 characters. If you change a form's title, remember to use the new title when you want to attach anything to that form. For more information, see the description of **attach**.

Example

{button ,AL(`OPAL_TYPE_FORM;OPAL_TYPE_APPLICATION;OPAL_METH_FOGETT;OPAL_METH_FOATTA;;',0, "Defaultoverview",)} Related Topics

setTitle example

See the **openAsDialog** example.

show method/procedure

Displays a minimized window at its previous size; makes a hidden form visible.

Syntax

```
show ( )
```

Description

show makes a hidden form visible. **show** also restores a minimized window to the size it was before it was minimized. This method is similar to the Restore command on the Control menu.

show doesn't make a form the top window; use **bringToTop** to make a form the top layer and give it focus.

Example

{button ,AL(`OPAL_TYPE_FORM;OPAL_METH_FOHIDE;OPAL_METH_FOIVIS;OPAL_TYPE_APPLICATION;',0,"Defaultoverview",)} **Related Topics**

show example

See the **hide** example.

showToolbar procedure

Makes the standard Toolbar visible.

Syntax

```
showToolbar ( )
```

Description

showToolbar displays the standard Toolbar.

Example

{button ,AL(`OPAL_TYPE_FORM;OPAL_METH_FOHTOOLBAR;OPAL_METH_FOISBARSHO;',0,"Defaultoverview",)} **Related Topics**

showToolbar example

See the **hideToolbar** example.

wait method

Suspends execution of a method.

Syntax

```
wait ( ) AnyType
```

Description

wait suspends execution of the current method until the form you're waiting for returns (see **formReturn**). This method is used to open a second form as a dialog box. Execution resumes in the first form when the second form (the one you're waiting for) calls **formReturn** or when the second form closes. After the called form returns, the calling form should close it with **close**. The called form does not automatically close, even if the user closes it; it stays open to allow the code on the calling form to examine the it (e.g., to see settings on a dialog box).

Example

{button ,AL(`OPAL_TYPE_FORM;OPAL_METH_FOFORR;OPAL_METH_FOFORMCALLER;OPAL_METH_FOOPAD;',0,"Defaultoverview",)} **Related Topics**

wait example

See the **formReturn** example.

windowClientHandle method/procedure

Returns the handle of a window.

Syntax

windowClientHandle ( ) LongInt

Description

A window handle is a unique integer identifier that is assigned to a window by Windows. **windowClientHandle** returns an integer value that represents the window handle of the client area of a form. When called as a procedure, it returns the window handle of the client area of the current form. This method should be used only by advanced programmers.

This information is useful only if you're using functions from a dynamic link library (DLL).

Example

{button ,AL(`OPAL_TYPE_FORM;OPAL_TYPE_APPLICATION;OPAL_METH_FOWINDOWHANDLE;;;',0,"Defaultoverview",)} **Related Topics**

windowClientHandle example

In the following example, assume that a dynamic link library (DLL) called MYTEST.DLL exists and that it contains a function called *doSomething*. The *doSomething* function takes one argument, a window handle. Because *doSomething* is not an ObjectPAL method, information about the method must be declared in the Uses window. The following code defines the prototype information for *doSomething* and appears in the Uses window at the form level:

```
;Form1::Uses

Uses MYTEST

  doSomething(wHandle CLONG)

EndUses
```

The following code appears in the **pushButton** method on the form:

```
; someButton::pushButton
method pushButton(var eventInfo Event)
doSomething(windowClientHandle())  ; call doSomething and supply the
                                   ; handle of the client portion
                                   ; of the current form
endMethod
```

windowHandle method/procedure

Returns the handle of a window.

Syntax

windowHandle ( ) LongInt

Description

A window handle is a unique integer identifier that is assigned to a window by Windows. **windowHandle** returns an integer value that represents the window handle of a form. When called as a procedure, **windowHandle** returns the window handle of the current form. This method should be used only by advanced programmers.

This information is useful only if you're using functions from a dynamic link library (DLL).

Example

{button ,AL(`OPAL_TYPE_FORM;OPAL_METH_FOWINDOWCLIENTHANDLE;',0,"Defaultoverview",)}

Related Topics

windowHandle example

In the following example, assume that a (DLL) called MYTEST.DLL exists and that it contains a function called *doSomething*. The *doSomething* function takes one argument, a window handle. Because *doSomething* is not an ObjectPAL method, information about the method must be declared in the Uses window. The following code defines the prototype information for *doSomething* and appears in the Uses window at the form level:

```
;Form1::Uses

Uses MYTEST

  doSomething(wHandle CLONG)

EndUses
```

The following code appears in the **pushButton** method on the form:

```
; someButton::pushButton
method pushButton(var eventInfo Event)
   doSomething(windowHandle())  ; call doSomething and supply the
                                ; window handle of the current form
endMethod
```

writeText method

Writes text contents of a form to file.

Syntax

```
writeText (const filename    String ) Logical
```

Description

writeText writes all text displayed on a form to the disk file specified by *filename*. This method attempts to keep the relative position of all text constants within the text file, but does not write the character or point size attributes. For forms with multiple pages, all text is written to the file, with the latter pages appended to the bottom of the file. This method writes only text. It does not write chart, graphic or OLE field information to the file.

Example

{button ,AL(`OPAL_TYPE_FORM;OPAL_METH_UIENUMSOURCETOFILE;OPAL_METH_GRWFILE;OPAL_METH_OLWRITETO;;',0,"Defaultoverview",)} Related Topics

writeText example

The following example writes the contents of the form BIOLIFE.FSL to the text file test.txt . This example assumes that a form named BIOLIFE.FSL already exists in the working directory.

```
method pushButton(var eventInfo Event)
 var
 f form
 endvar

if not f.open("BIOLIFE")  then ;// attempts to open Biolife.fsl. If not successful, alerts t
                                                    ;// the user and returns to the form
msginfo("stop","could not open Biolife form")
return
endif
 f.attach("BIOLIFE")                   ;// attaches form variable to biolife.fsl
 f.writetext(":WORK:test.txt")     ;// writes the displayed contents of biolife.fsl to the file
test.txt
 f.close()                                    ;// closes  biolife.fsl
endMethod
```

Graphic type

A Graphic variable provides a handle that is used to manipulate a graphic object. That is, you can use Graphic variables in ObjectPAL code to manipulate graphic objects. Graphic objects contain and display graphics in bitmap format (BMP). However, Corel Paradox can import the following graphic formats: bitmap (BMP), encapsulated Postscript (EPS), graphic interchange format (GIF), Paintbrush (PCX), and tagged information file format (TIF).

You can use Graphic type methods **readFromClipboard**, **writeToClipboard**, **readFromFile,** and **writeToFile** to transfer bitmaps between forms (and reports), tables, the Clipboard, and disk files.

The Graphic type includes several derived methods from the AnyType type.

Methods for the Graphic type

AnyType	**Graphic**
blank	**readFromClipboard**
dataType	**readFromFile**
isAssigned	**writeToClipboard**
isBlank	**writeToFile**
isFixedType	

Print related ObjectPAL methods and examples

readFromClipboard method

Reads a graphic from the Clipboard.

Syntax

```
readFromClipboard ( ) Logical
```

Description

readFromClipboard reads a graphic from the Clipboard to a variable of type Graphic. If the Clipboard contains a graphic that can be copied to the Graphic variable, **readFromClipboard** returns True. If the Clipboard is empty or does not contain a valid graphic, **readFromClipboard** returns False. **readFromClipboard** can read bitmap (BMP) and device independent bitmap (DIB) formats.

Example

{button ,AL(`OPAL_TYPE_GRAPHIC;OPAL_METH_GRRFILE;OPAL_METH_GRWCLIP;',0,"Defaultoverview",)}

Related Topics

readFromClipboard example

In the following example, a form contains a multi-record object named *BIOLIFE* bound to the *Biolife* table, and a button named *getGraphic*. The **pushButton** method for *getGraphic* locates the record with a Common Name field value of Firefish and writes the contents of the Clipboard to that record's Graphic field. If the Clipboard is empty or does not contain a graphic, the **readFromClipboard** method returns False and the value of the Graphic field is not changed.

```
; getGraphic::pushButton
method pushButton(var eventInfo Event)


var
  myGraphic Graphic
endVar


if BIOLIFE.locate("Common Name", "Firefish") then


  if myGraphic.readFromClipboard() then
    ; get the current Clipboard contents to myGraphic
    BIOLIFE.edit()                ; start Edit mode on the table
    BIOLIFE.Graphic = myGraphic   ; write the bitmap to the field
    BIOLIFE.endEdit()             ; end Edit mode
  endIf
endIf
endMethod
```

readFromFile method

Reads a graphic from a file.

Syntax

```
readFromFile ( const fileName String ) Logical
```

Description

readFromFile reads a graphic from a disk file specified in *fileName*. **readFromFile** returns True if the *fileName* name exists and contains a graphic format that can be imported; otherwise, it returns False. Corel Paradox can import the following graphic formats:

- bitmap (BMP)
- encapsulated Postscript (EPS)
- graphic interchange format (GIF)
- Paintbrush (PCX)
- tagged information file format (TIF)

Example

{button ,AL(`OPAL_TYPE_GRAPHIC;OPAL_METH_GRRCLIP;OPAL_METH_GRWFILE;',0,"Defaultoverview",)}

Related Topics

readFromFile example

The following example assumes that a form contains a button named *getChess* and an unbound graphic field named *bitmapField*. The **pushButton** method for *getChess* attempts to read the bitmap file CHESS.BMP from the C:\WINDOWS folder and stores CHESS.BMP in the *chessBmp* variable. If **readFromFile** is successful, *chessBmp* is written to the *bitmapField* object.

```
; getChess::pushButton
method pushButton(var eventInfo Event)
var
  chessBmp Graphic
endVar
; get the bitmap chess.bmp from the C:\Windows folder,
; and write it to the bitmapField graphic
if chessBmp.readFromFile("c:\\windows\\chess.bmp") then
  bitmapField = chessBmp
endIf
endMethod
```

writeToClipboard method

Writes a bitmap to the Clipboard.

Syntax

```
writeToClipboard ( ) Logical
```

Description

writeToClipboard writes a bitmap to the Clipboard. **writeToClipboard** returns True if successful; otherwise, it returns False. Formats copied to the Clipboard can be bitmap (BMP) or device independent bitmap (DIB).

Example

{button ,AL(`OPAL_TYPE_GRAPHIC;OPAL_METH_GRWFILE;OPAL_METH_GRRCLIP;',0,"Defaultoverview",)}

Related Topics

writeToClipboard example

The following example assumes that a form contains a button named *getChessToClip* and a bitmap field named *bitmapField*. The **pushButton** method for *getChessToClip* stores the value of *bitmapField* to *chessBmp* and then writes *chessBmp* to the Clipboard.

```
; getChessToClip::pushButton
method pushButton(var eventInfo Event)
var
   chessBmp Graphic
endVar
; get the bitmap from the bitmapField,
; and write it to the Clipboard
if NOT bitmapField.isblank() then
  chessBmp = bitmapField
  chessBmp.writeToClipboard()
endif
endMethod
```

writeToFile method

Writes a bitmap to a file.

Syntax

```
writeToFile ( const fileName String ) Logical
```

Description

writeToFile writes a bitmap to a disk file specified in *fileName*. If *fileName* does not specify a path, this method writes to the working directory (:WORK:). **writeToFile** returns True if the file specified can be created; otherwise, it returns False.

Example

{button ,AL(`OPAL_TYPE_GRAPHIC;OPAL_METH_GRWCLIP;',0,"Defaultoverview",)} Related Topics

writeToFile example

The following example assumes that a form contains a button named *writeChessToFile* and a bitmap named *bitmapField*. The **pushButton** method for *writeChessToFile* stores the value of *bitmapField* to *chessBmp* and then writes *chessBmp* to a file named CHESS1.BMP in the working directory.

```
; writeChessToFile::pushButton
method pushButton(var eventInfo Event)
var
   chessBmp Graphic
endVar
; get the bitmap from the bitmapField,
; and write it to the Clipboard
if NOT bitmapField.isblank() then
  chessBmp = bitmapField
  chessBmp.writeToFile("chess1.bmp")
endif
endMethod
```

KeyEvent type

A KeyEvent object gets and sets information about keystroke events.

The following built-in event methods are triggered by the KeyEvents **keyChar** and **keyPhysical**.

The KeyEvent type includes several derived methods from the Event type.

Methods for the KeyEvent type

Event	**KeyEvent**
errorCode	**char**
getTarget	**charAnsiCode**
isFirstTime	**isAltKeyDown**
isPreFilter	**isControlKeyDown**
isTargetSelf	**isFromUI**
reason	**isShiftKeyDown**
setErrorCod	**setAltKeyDown**
setReason	**setChar**
	setControlKeyDown
	setShiftKeyDown
	setVChar
	setVCharCode
	vChar
	vCharCode

Print related ObjectPAL methods and examples

char method

Returns the character associated with a keystroke.

Syntax

```
char ( ) String
```

Description

char returns the character associated with a keystroke. For example, if you type a, **char** returns a. If you press SHIFT + A, **char** returns A. If a keystroke results in an unprintable character, **char** returns an empty string ("").

char is the easiest way to check for an alphanumeric keystroke when case matters. If case doesn't matter, use **vChar** to test against the string value of a virtual key code. For example, if it matters whether the user presses a lowercase a or an uppercase A, use **char** to return the string value of the character pressed, and compare it to a or A. If you want to find out if either a or A was pressed, use **vChar** and compare it to A (the virtual key code string for either a lowercase a or an uppercase A).

Example

{button ,AL(`OPAL_TYPE_KEYEVENT;OPAL_METH_KECHARANSICODE;OPAL_METH_KESCHA;OPAL_METH_KEVCHA;OPAL_METH_STANSICODE;OPAL_METH_STCHRTOKEYNAME;OPAL_METH_STTOANSI;OPAL_METH_STTOOEM;',0,"Defaultoverview",)} **Related Topics**

char example

The following example displays the character typed into a field object as a message at the bottom of the screen. The code is attached to a field object's built-in **keyChar** method.

```
; thisField::keyChar
method keyChar(var eventInfo KeyEvent)
  doDefault                  ; put character in the field
  message(eventInfo.char())  ; then display character as a message
endMethod
```

charAnsiCode method

Returns the ANSI value associated with a keystroke.

Syntax

```
charAnsiCode ( ) SmallInt
```

Description

charAnsiCode returns an integer that represents the ANSI value associated with a keystroke. For example, if you type a, **charAnsiCode** returns 97. If you press SHIFT + A, **charAnsiCode** returns 65. **charAnsiCode** works with unprintable characters as well. For example, if you press ENTER, **charAnsiCode** returns 13.

Example

{button ,AL(`OPAL_TYPE_KEYEVENT;OPAL_METH_KECHAR;OPAL_METH_KEVCHA;OPAL_METH_KESVCH;OPAL_METH_STANSICODE;OPAL_METH_STCHRTOKEYNAME;OPAL_METH_STTOANSI;OPAL_METH_STTOOEM;',0,"Defaultoverview",)} Related Topics

charAnsiCode example

The following example beeps when a user presses BACKSPACE or CTRL + H. This code is attached to a field object's built-in **keyPhysical** method.

```
; thisField::keyPhysical
method keyPhysical(var eventInfo KeyEvent)
if eventInfo.charAnsiCode() = 8 then  ; if user presses CTRL + H or BACKSPACE
  beep()                              ; make a sound
endif
endMethod
```

isAltKeyDown method

Reports whether ALT was held down during a KeyEvent.

Syntax

```
isAltKeyDown ( ) Logical
```

Description

isAltKeyDown returns True if ALT was held down at the time a KeyEvent occurred; otherwise, it returns False.

Example

{button ,AL(`OPAL_TYPE_KEYEVENT;OPAL_METH_KEICKD;OPAL_METH_KEISKD;OPAL_METH_KESETALTKEYDOWN;',0,"Defaultoverview",)} Related Topics

isAltKeyDown example

The following example assumes a form has a box named *boxOne*. When the user presses ALT + C, the **keyPhysical** method for the form changes the color of *boxOne*. This code is attached to a form's **keyPhysical** method

```
; thisForm::keyPhysical
method keyPhysical(var eventInfo KeyEvent)
if eventInfo.isPreFilter()
  then
    ;code here  executes for each object in form


    if eventInfo.isAltKeyDown() AND    ; if user presses ALT + C
       eventInfo.vChar() = "C" then
       disableDefault                  ; block normal processing
       ; alternate a boxOne's color between red and blue
       boxOne.color = iif(boxOne.color = Red, Blue, Red)
    endif


  else
    ;code here executes just for form itself
endif
endMethod
```

isControlKeyDown method

Reports whether CTRL was held down during a KeyEvent.

Syntax

```
isControlKeyDown ( ) Logical
```

Description

isControlKeyDown returns True if CTRL was held down at the time a KeyEvent occurred; otherwise, it returns False.

Example

{button ,AL(`OPAL_TYPE_KEYEVENT;OPAL_METH_KESETCONTROLKEYDOWN;OPAL_METH_KEISALTKEYDOWN;OPAL_METH_KEISKD;',0,"Defaultoverview",)} Related Topics

isControlKeyDown example

See the **setControlKeyDown** example.

isFromUI method

Reports whether an event was generated by the user interacting with Corel Paradox.

Syntax

isFromUI ( ) Logical

Description

isFromUI reports whether a KeyEvent was generated either by the user interacting with Corel Paradox or internally (e.g., by an ObjectPAL statement). This method returns True only for the first KeyEvent generated by a keystroke; for subsequent events and actions, it returns False.

Example

{button ,AL(`OPAL_TYPE_KEYEVENT;OPAL_METH_EVISPREFILTER;',0,"Defaultoverview",)} Related Topics

isFromUI example

The following example shows how to put one of two messages on the Status Bar depending on whether a character is put in a field by a user or by ObjectPAL. This method returns True for user actions (including **sendKeys**, which mimics user input). It returns False with all other ObjectPAL methods, including **keyPhysical**.

The following code is attached to the **pushButton** method of a button named *btnAutoFill*. This method sends the character a to the field *fldPassword*:

```
; btnAutofill :: pushButton
method pushButton(var eventInfo Event)
   fldPassword.keyPhysical(97, 97, Shift)   ; send an "a"
endMethod
```

The following code is attached to the **keyPhysical** method of a field named *fldPassword*. This method sends one of two messages depending on whether the user typed in a character or used the *btnAutofill* button:

```
;fldPassword :: keyPhysical
method keyPhysical(var eventInfo KeyEvent)
   if eventInfo.isFromUI() then
      message("Try using the autofill button.")
   else
      message("Automatically typing value.")
   endIf
endMethod
```

isShiftKeyDown method

Reports whether SHIFT was held down during a KeyEvent.

Syntax

```
isShiftKeyDown ( ) Logical
```

Description

isShiftKeyDown returns True if SHIFT was held down at the time a KeyEvent occurred; otherwise, it returns False.

Example

{button ,AL(`OPAL_TYPE_KEYEVENT;OPAL_METH_KESSKD;OPAL_METH_KEISALTKEYDOWN;OPAL_METH_KEICKD;',0,"Defaultoverview",)} Related Topics

isShiftKeyDown example

See the **setShiftKeyDown** example.

setAltKeyDown method

Simulates pressing and holding ALT during a KeyEvent.

Syntax

```
setAltKeyDown ( const yesNo Logical )
```

Description

setAltKeyDown adds information about the state of ALT to a KeyEvent. You must specify Yes or No. Yes means ALT was pressed during a KeyEvent; No means ALT was not pressed.

Example

{button ,AL(`OPAL_TYPE_KEYEVENT;OPAL_METH_KEISALTKEYDOWN;OPAL_METH_KESETCONTROLKEYDOWN;OPAL_METH_KESSKD;',0,"Defaultoverview",)} **Related Topics**

setAltKeyDown example

The following example assumes a form has a box named *boxOne*. When the user presses ALT + C, the **keyPhysical** method for the form changes the color of *boxOne*. This code is attached to a form's **keyPhysical** method:

```
; thisForm::keyPhysical
method keyPhysical(var eventInfo KeyEvent)
if eventInfo.isPreFilter()
  then
    ; code here executes for each object in form
    if eventInfo.isAltKeyDown() and    ; if user presses ALT + C
       eventInfo.vChar() = "C" then
       disableDefault                  ; block normal processing
       ; alternate a boxOne's color between red and blue
       boxOne.color = iif(boxOne.color = Red, Blue, Red)
    endif
  else
    ; code here executes just for form itself
endif
endMethod
```

To simulate pressing ALT + C, the code for this method creates a KeyEvent variable and sets its virtual key character to C and sets the ALT key down.

```
; sendAltC::pushButton
method pushButton(var eventInfo Event)
var
  ke KeyEvent
endVar
ke.setVChar("C")             ; set the character to C
ke.setAltKeyDown(Yes)        ; set the ALT key state to pressed
thisForm.keyPhysical(ke)     ; send off the event
endMethod
```

setChar method

Specifies an ANSI character for a KeyEvent.

Syntax

```
setChar ( const char String )
```

Description

setChar sets a KeyEvent to have an ANSI character based on the value of *char*, where *char* evaluates to single character string (e.g., a).

Example

{button ,AL(`OPAL_TYPE_KEYEVENT;OPAL_METH_KESVCH;OPAL_METH_KESETVCHARCODE;OPAL_METH_STTOOEM;OPAL_METH_STCHR;OPAL_METH_STCHROEM;OPAL_METH_STKEYNAMETOCHR;OPAL_METH_STTOANSI;',0,"Defaultoverview",)} **Related Topics**

setChar example

The following example attaches code to a field's built-in **keyChar** method. The **keyChar** method for *fieldOne* converts each space to an underscore as the user types characters into the field.

```
; thisField::keyChar
method keyChar(var eventInfo KeyEvent)
  if eventInfo.Char() = " " then  ; when user enters a space
    eventInfo.setChar("_")        ; convert it to underscore
  endif                           ; process other keystrokes normally
endMethod
```

setControlKeyDown method

Simulates pressing and holding CTRL during a KeyEvent.

Syntax

```
setControlKeyDown ( const yesNo Logical )
```

Description

setControlKeyDown adds information about the state of CTRL to **eventInfo** for a KeyEvent. You must specify Yes or No. Yes means CTRL was pressed during a KeyEvent; No means CTRL was not pressed.

Example

{button ,AL(`OPAL_TYPE_KEYEVENT;OPAL_METH_KEICKD;OPAL_METH_KESETALTKEYDOWN;OPAL_METH_KESSKD;',0,"Defaultoverview",)} Related Topics

setControlKeyDown example

The following example assumes a form has a box named *boxOne*. When the user presses CTRL + C, the **keyPhysical** method for the form changes the color of *boxOne*. This code is attached to a form's **keyPhysical** method:

```
; thisForm::keyPhysical
method keyPhysical(var eventInfo KeyEvent)
if eventInfo.isPreFilter() then
  ; code here executes for each object in form
  if eventInfo.isControlKeyDown() and   ; if user presses CTRL + C
    eventInfo.vChar() = "C" then
    disableDefault                      ; block normal processing
    ; alternate color of boxOne between red and blue
    boxOne.color = iif(boxOne.color = Red, Blue, Red)
  endif
else
  ; code here executes just for form itself
endif
endMethod
```

To simulate CTRL + C, the code for this method creates a KeyEvent variable and sets its virtual key character to C and sets the CTRL key down.

```
; sendCTRLC::pushButton
method pushButton(var eventInfo Event)
var
  ke KeyEvent
endVar
ke.setChar("C")              ; set the character to C
ke.setControlKeyDown(Yes)    ; set the CTRL key state to pressed
thisForm.keyPhysical(ke)     ; send off the event
endMethod
```

setShiftKeyDown method

Simulates pressing and holding SHIFT during a KeyEvent.

Syntax

```
setShiftKeyDown ( const yesNo Logical )
```

Description

setShiftDown adds information about the state of SHIFT to a KeyEvent. You must specify Yes or No. Yes means SHIFT was pressed and held; No means SHIFT wasn't pressed.

Example

{button ,AL(`OPAL_TYPE_KEYEVENT;OPAL_METH_KEISKD;OPAL_METH_KESETALTKEYDOWN;OPAL_METH_KESETCONTROLKEYDOWN;',0,"Defaultoverview",)} Related Topics

setShiftKeyDown example

The following example assumes a form has a box named *boxOne*. When the user presses SHIFT + C, the **keyPhysical** method for the form changes the color of *boxOne*. This code is attached to a form's **keyPhysical** method:

```
; thisForm::keyPhysical
method keyPhysical(var eventInfo KeyEvent)
if eventInfo.isPreFilter() then
  ; code here executes for each object in form
  if eventInfo.isShiftKeyDown() and    ; if user presses CTRL + C
    eventInfo.vChar() = "C" then
    disableDefault                     ; block normal processing
    ; alternate color of boxOne between red and blue
    boxOne.color = iif(boxOne.color = Red, Blue, Red)
  endif
else
  ; code here executes just for form itself
endif
endMethod
```

To simulate pressing SHIFT + C, the code for this method creates a KeyEvent variable, sets its virtual key character to C, and sets the SHIFT key down.

```
; sendShiftC::pushButton
method pushButton(var eventInfo Event)
var
  ke KeyEvent
endVar
ke.setVChar("C")           ; set the character to C
ke.setShiftKeyDown(Yes)    ; set the SHIFT key state to pressed
thisForm.keyPhysical(ke)   ; send off the event
endMethod
```

setVChar method

Specifies a Windows virtual character for a KeyEvent.

Syntax

```
setVChar ( const char String )
```

Description

setVChar specifies in *char* a one-character string for a KeyEvent. Use **setVChar** with an uppercase letter or a Keyboard constant to specify a code string for a single letter, but use the constant as a quoted string instead of an integer value. In the following example, the code statement specifies a tab character:

```
eventInfo.setVChar("VK_TAB")
```

The virtual character code string for any letter is the uppercase letter. For example, the virtual character code string for the letter k is K (uppercase only).

Example

{button ,AL(`OPAL_TYPE_KEYEVENT;OPAL_METH_KESCHA;OPAL_METH_KESETVCHARCODE;OPAL_METH_STCHR;OPAL_METH_STCHROEM;',0,"Defaultoverview",)} **Related Topics**

setVChar example

See the **setAltKeyDown** example or the String type chrToKeyName.

setVCharCode method

Specifies a Windows virtual character for a KeyEvent.

Syntax

```
setVCharCode ( const VK_Constant SmallInt )
```

Description

setVCharCode uses a Keyboard constant in *VK_Constant* to specify a Windows virtual character for a KeyEvent.

Example

{button ,AL(`OPAL_TYPE_KEYEVENT;OPAL_METH_KESCHA;OPAL_METH_KESVCH;OPAL_METH_STKEYNAMETOVKCODE;',0,"Defaultoverview",)} Related Topics

setVCharCode example

The following example attaches code to a form's built-in **keyPhysical** method. When the user types ?, this code invokes the Corel Paradox Help system:

```
; thisForm::keyPhysical
method keyPhysical(var eventInfo KeyEvent)
if eventInfo.isPreFilter()
  then
    ; code here executes for each object in form
    if eventInfo.char() = "?" then   ; if user types ?
      eventInfo.setVCharCode(VK_HELP)  ; invoke built-in help system
    endif
  else
    ; code here executes just for form itself
endif
endMethod
```

vChar method

Returns a Windows virtual character.

Syntax

```
vChar ( ) String
```

Description

vChar returns a Windows virtual key name as a string. Use Keyboard constants to find out which Windows virtual character was returned, but use the constants as quoted strings instead of integer values. In the following example, the statements are equivalent (they both beep when you press Return). The first statement uses **vCharCode** and the constant VK_RETURN to test for an integer value, the second statement uses **vChar** and VK_RETURN to test for a string value.

```
if vCharCode = VK_RETURN then beep() endIf
if vChar = "VK_RETURN" then beep() endIf
```

Example

{button ,AL(`OPAL_TYPE_KEYEVENT;OPAL_METH_KECHAR;OPAL_METH_STANSICODE;OPAL_METH_STCHRTOKEYNAME;OPAL_METH_STTOANSI;OPAL_METH_STTOOEM;',0,"Defaultoverview",)} Related Topics

vChar example

In the following example, assume a form contains a box named *boxOne*. When the user presses a movement key, this code moves *boxOne* in increments of 100 twips. If SHIFT is held down in combination with a movement key, *boxOne* moves 1000 twips. Because **vChar** returns the virtual key name as a string, this code must compare key names against string values such as VK_LEFT. This code is attached to a form's built-in **keyPhysical** method:

```
; thisForm::keyPhysical
method keyPhysical(var eventInfo KeyEvent)
var
  kp      String        ; key name of the keystroke
  posPt   Point         ; x and y position of the box object
  boxStep SmallInt      ; number of Points to move the box
  x, y    LongInt       ; coordinates of the box object
endVar


if eventInfo.isPreFilter()
  then
    ;code here executes for each object in form
    disableDefault                  ; don't execute built-in code


    kp = eventInfo.vChar()          ; load kp with vChar string
    posPt = boxOne.position         ; posPt stores current position of box
    x = posPt.x()                   ; x stores the horizontal position
    y = posPt.y()                   ; y stores the vertical position


    ; if the SHIFT key was held down when the movement key was pressed,
    ; assign a large number to boxStep, else, a small number
    boxStep = iif(eventInfo.isShiftKeyDown(), 1000, 100)


    ; this block assigns x or y variables according to
    ; the key combination that the user presses
    switch
      case kp = "VK_LEFT"  : x = x - boxStep
      case kp = "VK_RIGHT" : x = x + boxStep
      case kp = "VK_UP"    : y = y - boxStep
      case kp = "VK_DOWN"  : y = y + boxStep
      otherwise            : enableDefault    ; let built-in code execute
    endswitch


    ; now move the box to location specified by x and y variables,
    ; and display the virtual key name associated with the keystroke
    boxOne.position = Point(x,y)
    message("Value of vChar() was " + kp)


  else
    ;code here executes just for form itself
endif
endMethod
```

vCharCode method

Returns the integer value of a Windows virtual character.

Syntax

vCharCode ( ) SmallInt

Description

vCharCode returns the integer value of a Windows virtual character. Use Keyboard constants to find out which Windows virtual character the integer value represents.

Example

{button ,AL(`OPAL_TYPE_KEYEVENT;OPAL_METH_KEVCHA;',0,"Defaultoverview",)} Related Topics

vCharCode example

For the following example, assume a form has a field named *thisField*. When the user types a value in *thisField* and presses Return, the code creates and executes a query based on the value of the field. This code is attached to the built-in **keyPhysical** method for *thisField*.

```
; thisField::keyPhysical
method keyPhysical(var eventInfo KeyEvent)
var
  cName String      ; used as tilde var
  qVar Query          ; the query statement
  tv TableView      ; tableView handle
endVar


if eventInfo.vCharCode() = VK_RETURN then  ; if user presses Enter
  cName = self.value                          ; store value of field
  qVar = Query

         c:\Corel\Suite8\Paradox\samples\biolife.db|Common Name |Species Name|
                                    |check ~cName|check|


       endQuery

  ; run query, write contents to myFish table
  qVar.executeQBE("myFish.db")
  tv.open("myFish")              ; view myFish view
endif
endMethod
```

Library type

A library is a Corel Paradox object that stores custom methods, custom procedures, variables, constants, and user-defined data types. Libraries are used to store and maintain frequently-used routines and to share custom methods and variables among several forms.

In many ways, working with a library is like working with a form. Like a form, a library has built-in event methods. You add code to a library, just as you do to a form, by using the Object Explorer and the ObjectPAL Editor. (However, you can't place design objects in the library.) As with a form, you can open Editor windows to declare custom ObjectPAL methods, procedures, variables, constants, data types, and external routines.

The Library type includes several derived methods from the Form type.

Methods for the Library type

Form	**Library**
deliver	**close**
isCompileWit	**create**
load	**enumSource**
methodDelet	**enumSourceToFile**
methodGet	**execMethod**
methodSet	**methodEdit**
save	**open**
setCompileW	

Print related ObjectPAL methods and examples

close method

Closes a library.

Syntax

```
close ( )
```

Description

close closes a library and ends the association between a Library variable and the underlying library file.

Example

{button ,AL(`OPAL_TYPE_LIBRARY;OPAL_METH_LBOPEN;',0,"Defaultoverview",)} **Related Topics**

close example

The following example declares a Library variable named *lib*, and calls **open** to associate *lib* with the library TOOLS.LSL. The example executes a method from that library and then calls **close** to end the association between the variable and the library. Another call to **open** associates *lib* with the library KIT.LSL to make methods in that library available.

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
  lib Library          ; declare a Library variable
endVar


lib.open("TOOLS.LSL")  ; associate lib with the library TOOLS.LSL
lib.doThis()           ; execute a method from the library
lib.close()            ; end the association between lib and the library


lib.open("KIT.LSL")    ; associate lib with another library
lib.doThat()           ; execute a method from the library


endMethod
```

create method

Creates a library.

Syntax

create ( ) Logical

Description

create creates a blank library and leaves it in a design window. You can use **methodSet** (derived from the Form type) to alter or add methods in the new library.

Example

{button ,AL(`OPAL_TYPE_LIBRARY;OPAL_METH_LBOPEN;',0,"Defaultoverview",)} **Related Topics**

create example

The following example uses **create** to make a new library, adds a custom method to it with **methodSet**, save the library with **save** and then close the library.

```
; btnCreateLibrary :: pushButton
method pushButton(var eventInfo Event)
   var
      lib   Library
   endVar


   ;Create library.
   lib.create()
   lib.methodSet("cmMessage", "method cmMessage()
       msgInfo(\"From new library\", \"Hello World!\") endMethod")
   lib.save("library")
   lib.close()


endmethod
```

enumSource method

Writes the code from a library to a Corel Paradox table.

Syntax

```
enumSource ( const tableName String [ , const recurse Logical ] )
```

Description

enumSource lists, in the Corel Paradox table specified in *tableName,* all the custom code (e.g., methods, procedures, and variables) stored in a library. If the table does not exist, Corel Paradox creates it in the working directory; if the table does exist, information is appended to the table.

The structure of the table is:

Field name	Type	Size
Object	A	128
MethodName	A	128
Source	M	64

The Object field stores the UIObject name of the library, the MethodName field stores the name of the method, procedure, or window (Var, Const, Proc, Type, or Uses), and the Source field stores the corresponding source code.

This method also applies to the Form type. For forms, the optional argument *recurse* specifies whether to include overridden methods for all objects contained by the form. Because a Library does not contain objects, the *recurse* argument is not meaningful in the context of a Library.

You must **open** or **load** the library before calling this method.

Example

{button ,AL(`OPAL_TYPE_LIBRARY;OPAL_METH_LBENUMSOURCETOFILE;OPAL_METH_LBOPEN;',0,"Defaultoverview",)} **Related Topics**

enumSource example

The following example declares a Library variable named *lib*, calls **open** to associate *lib* with the library TOOLS.LSL, and calls **enumSource** to list the code from the library to a Corel Paradox table named LIBSRC.DB:

```
; srcToTable::pushButton
method pushButton(var eventInfo Event)
var
  lib Library
endVar

if lib.open("TOOLS.LSL", PrivateToForm) then

  ; write contents of TOOLS.LSL to LIBSRC.DB--
  ; goes to :WORK: by default
  lib.enumSource("LIBSRC.DB")

else
  msgStop("TOOLS.LSL", "Could not open library.")
endIf

endMethod
```

enumSourceToFile method

Writes the code from a library to a text file.

Syntax

enumSourceToFile (const ***fileName*** String [**,** const ***recurse*** Logical] **)**

Description

enumSourceToFile lists all the custom code (e.g., methods, procedures, and variables) stored in a library to the text file specified in *fileName*. If the file does not exist, Corel Paradox creates it. If the file does exist, Corel Paradox overwrites it without asking for confirmation. If *fileName* contains no path or alias, the file is created in the working directory.

In the text file, comment lines are used to identify and mark the beginning and end of each method, procedure, or variable. The following example shows the code for a library's built-in **open** method:

```
;|BeginMethod|#Library1|open|
method open(var eventInfo Event)
   var
      myMsgTCursor   Tcursor
   endVar
   if not myMsgCursor.open("Msghelp.db") then
      msgStop("Error", "Couldn't open MsgHelp.db")
      fail()
   endIf
endMethod
;|EndMethod|#Library1|open|
```

This method also applies to the Form type. For forms, the optional argument *recurse* specifies whether to include overridden methods for all objects contained by the form. Because a Library does not contain objects, the *recurse* argument is not meaningful in the context of a Library.

You must call **open** or **load** the library before calling this method.

Example

{button ,AL(`OPAL_TYPE_LIBRARY;OPAL_METH_LBENUMSOURCE;OPAL_METH_LBOPEN;',0,"Defaultoverview",)} Related Topics

enumSourceToFile example

The following example declares a Library variable named *lib*, calls **open** to associate *lib* with the library TOOLS.LSL, and calls **enumSourceToFile** to list the code from the library to a text file named LIBSRC.TXT.

```
; getSource::pushButton
method pushButton(var eventInfo Event)
var
  lib Library
endVar


if lib.open("TOOLS.LSL", PrivateToForm) then


  ; write contents of TOOLS.LSL to LIBSRC.TXT--
  ; goes to :PRIV: by default
  lib.enumSourceToFile("LIBSRC.TXT")


else
  msgStop("TOOLS.LSL", "Could not open library.")
endIf


endMethod
```

execMethod method

Calls a custom method that takes no arguments.

Syntax

```
execMethod ( const methodName String )
```

Description

execMethod calls the custom method indicated by the string *methodName*. The method named in *methodName* takes no arguments. **execMethod** allows you to call a library method based on the contents of a variable, which means the compiler does not know the method to call until run time.

Example

{button ,AL(`OPAL_TYPE_LIBRARY;OPAL_METH_LBOPEN;',0,"Defaultoverview",)} **Related Topics**

execMethod example

The following example creates an array of three items, each of which is the name of a custom method in a library. The code opens the library and calls **execMethod** for each item in the array:

```
var
   lib Library
   libMethods Array[3] String
   i SmallInt
endVar


libMethods[1] = "doThis"
libMethods[2] = "doThat"
libMethods[3] = "doOther"


if lib.open("tools.lsl", GlobalToDeskTop) then
   for i from 1 to libMethods.size()
      lib.execMethod(libMethods[i])
   endFor
else
   msgStop("TOOLS.LSL", "Could not open library.")
endIf
```

methodEdit method

Opens a library's method in an Editor window.

Syntax

```
methodEdit (const methodName String) Logical
```

Description

methodEdit opens the method specified by *methodName* in an Editor window. If you specify a method that doesn't exist, **methodEdit** will create it for you. **methodEdit** fails if you try to open a method that is running.

Example

methodEdit example

The following example opens the library's **testMethod** method in an editor window:

```
method pushButton(var eventInfo Event)
var
    MyLib  library
endvar
MyLib.load("Main.lsl")
MyLib.methodEdit("testMethod")
endMethod
```

open method

Associates a Library variable with a library and makes the library code available.

Syntax

```
open ( const libraryName String [ , const libScope SmallInt ] ) Logical
```

Description

open associates a Library variable with a library and makes the library code, variables, constants, and type declarations available to the form. Variables declared in the library can be kept private to the form, or they can be shared with other forms and libraries that have opened this library, depending on the value of *libScope*. ObjectPAL defines the following two LibraryScope constants to specify the scope of variables declared in the library:

- PrivateToForm specifies that each form that opens the library has its own copy of the variables.
- GlobalToDesktop specifies that every form in the desktop (Corel Paradox session) that opens the library shares the variables declared in the library.

To open a library and make its variables available to every form that opens the library in the current session of Corel Paradox, use the constant GlobalToDesktop. The following example opens the library MYLIB.LSL:

```
lib.open("myLib.lsl", GlobalToDesktop)
```

For two or more forms to share the same library, each form must open the library global to the desktop, and each form must have a Uses window that declares which library routines to use. This level of scope is useful in multiform applications because it allows several forms access to the same custom methods and allows the forms to share the same global variables.

A library can be opened private to the form in one form and global to the desktop in another form. Corel Paradox will load a new instance of the library, if necessary.

By default, a library opens global to the desktop. The following statements are equivalent:

```
lib.open("myLib.lsl") ; these statements are equivalent
lib.open("myLib.lsl", GlobalToDesktop)
```

Example

{button ,AL(`OPAL_TYPE_LIBRARY;OPAL_METH_LBCLOSE;',0,"Defaultoverview",)} Related Topics

open example

The following example shows how two forms can open a library global to the desktop and share the library. The following code is attached to a form's built-in **open** method, and opens *libOne* private to the form. *libOne* cannot be shared. *libTwo* is opened global to the desktop and can be shared. *libOne* and *libTwo* are library variables that have been declared in the **var** block of the form.

```
; formOne::open
method open(var eventInfo Event)


if eventInfo.isPreFilter()
   then
   ; code here executes for each object in the form
else
   ; code here executes just for the form itself


   libOne.open("TOOLS.LSL", PrivateToForm)   ; no sharing variables
                                             ; with other forms
   libTwo.open("KIT.LSL", GlobalToDesktop)   ; can be shared
                                             ; with other forms
endIf
endMethod
```

The following code is attached to another form's built-in **open** method. This code calls **open** to open the library KIT.LSL global to the desktop. This form and the previous form can now share KIT.LSL. *kitLib* is a library variable declared in the **var** block of the form.

```
; formTwo::open
method open(var eventInfo Event)


if eventInfo.isPreFilter()
   then
   ; code here executes for each object in the form
else
   ; code here executes just for the form itself
   kitLib.open("KIT.LSL", GlobalToDesktop) ; can be shared with other forms


endIf
endMethod
```

Logical type

Logical variables have two possible values: True or False. You can use the ObjectPAL constants Yes or On in place of True, and use No or Off in place of False.

A Logical variable occupies 1 byte of storage. In order of precedence, the logical operators are NOT, AND, and OR.

Logical variables often answer questions about other objects and operations, for example:

- Did that statement execute successfully?
- Is that table empty?
- Is that form displayed as an icon?

The Logical type includes several derived methods from the AnyType type.

Methods for the Logical type

Print related ObjectPAL methods and examples

logical procedure

Casts a value as type Logical.

Syntax

```
logical ( const value AnyType ) Logical
```

Description

logical casts *value* to the data type Logical. If *value* is a numeric data type, non-zero values evaluate to True and zero evaluates to False. If *value* is a string, it must evaluate to "True" or "False." (However, you can use True or False without the quotation marks.) ObjectPAL also provides Logical constants: On and Yes for True and Off and No for False.

Example

{button ,AL(`OPAL_TYPE_LOGICAL;',0,"Defaultoverview",)} **Related Topics**

logical example

In the following example, the **pushButton** method of a button named *showLogical* creates a string, casts it to a Logical type, then displays the result:

```
; showLogical::pushButton
method pushButton(var eventInfo Event)
var
  myVal     String
  theResult Logical
endVar
myVal = "True"             ; set a String of True
theResult = logical(myVal) ; and cast it to a Logical type
theResult.view()           ; show the result--Title displays Logical
endMethod
```

LongInt type

LongInt values are long integers; that is, they can be represented by a long series of digits. A LongInt variable occupies 4 bytes. ObjectPAL converts LongInt values to range from -2,147,483,648 to 2,147,483,647. The following example attempts to assign a value outside of this range to a LongInt variable causes an error:

```
var
   x, y, z LongInt
endVar


x = 2147483647 ; The upper limit value for a LongInt variable.
y = 1
z = x + y      ; This statement causes an error.
```

When ObjectPAL performs an operation on LongInt values, it expects the result to be a LongInt. That's why the addition operation in the previous example causes an error: the result is too large to be a LongInt. To work with a boundary value (in either the positive or negative direction), you must convert the value to a type that can accommodate it. In the following example, ObjectPAL converts one LongInt to a Number before doing the addition, and the statement succeeds. This example also assigns the result to a Number variable (which can handle the large value), instead of assigning it to a LongInt variable (which could not).

```
var
   x, y LongInt
   z    Number ; Declare z as a Number so it can hold the result.
endVar


x = 2147483647 ; The upper limit value for a LongInt variable.
y = 1
z = Number(x) + y ; This statement succeeds.
```

Note

- Run-time library methods defined for the Number type also work with LongInt variables. The syntax is the same, and the returned value is a number.

The following table displays the methods for the LongInt type, including several derived methods from the Number and AnyType types.

Methods for the LongInt type

AnyType	Number	**LongInt**
blank	abs	**bitAND**
dataType	acos	**bitIsSet**
isAssigned	asin	**bitOR**
isBlank	atan	**bitXOR**
isFixedType	atan2	**LongInt**
view	ceil	
	cos	
	cosh	
	exp	
	floor	
	fraction	
	fv	
	ln	
	log	
	max	
	min	
	mod	
	number	
	numVal	
	pmt	
	pow	
	pow10	

pv
rand
round
sin
sinh
sqrt
tan
tanh
truncate

Print related ObjectPAL methods and examples

bitAND method

Performs a bitwise AND operation on two values.

Syntax

```
bitAND ( const value LongInt ) LongInt
```

Description

bitAND returns the result of a bitwise AND operation on *value*. **bitAND** operates on the binary representations of two integers and compares them one bit at a time. The truth table for **bitAND** is:

a	b	a bitAND b
0	0	0
1	0	0
0	1	0
1	1	1

Example

{button ,AL(`OPAL_TYPE_LONGINT;OPAL_METH_LIBOR;OPAL_METH_LIBXOR;',0,"Defaultoverview",)}

Related Topics

bitAND example

In the following example, the **pushButton** method for a button named *andTwoNums* takes two integers and performs a bitwise AND calculation on them. The result of the calculation is displayed in a dialog box.

```
; andTwoNums::pushButton
method pushButton(var eventInfo Event)
var
  a, b LongInt
endVar
a = 33333   ; binary 00000000 00000000 10000010 00110101
b = -77777  ; binary 11111111 11111110 11010000 00101111
a.bitAND(b) ; binary 00000000 00000000 10000000 00100101
msgInfo("The result of a bitAND b is:", a.bitAND(b))
; displays 32805
endMethod
```

bitIsSet method

Reports whether a bit is 1 or 0.

Syntax

```
bitIsSet ( const value LongInt ) Logical
```

Description

bitIsSet examines the binary representation of an integer and reports whether the **value** bit is 0 or 1. **bitIsSet** returns True if the bit specified is 1 and False if the bit is 0.

value is a number specified by 2^n, where *n* is an integer between 0 and 30. The exponent *n* corresponds to one less than the position of the bit to test (counting from the right). For example, to specify the third bit from the right, use
$4(2^{(3-1)}$, which is 2^2).

Examples

{button ,AL(`OPAL_TYPE_LONGINT;OPAL_METH_LIBAND;OPAL_METH_LIBOR;OPAL_METH_LIBXOR;',0,"Defaultoverview",)} **Related Topics**

bitIsSet methods examples

Example1 Using **pushButton**

Example2 Using **bitIsSet** to display an integer as a binary number

bitIsSet example 1

In the following example, the **pushButton** method for a button named *isABitSet*, examines the values in two unbound field objects: *whichBit* and *whatNum*. *whichBit* contains the bit position (counting from the right) of the bit to test. *whatNum* contains the long integer to test.

The **pushButton** method uses *whichBit* to calculate the value of the position and assigns the result to *bitNum*. This method then checks *Num* to see if the *bitNum* bit is set, and displays the Logical result with a **msgInfo** dialog box:

```
; isABitSet::pushButton
method pushButton(var eventInfo Event)
var
  bitNum,
  Num      LongInt
endVar
; get the bit position number from the whichBit
; field and convert to multiple of 2
bitNum = LongInt(pow(2, whichBit - 1))
; get the number to test from the whatNum field
Num = whatNum
; is the bit for value bitNum 1 in Num?
msgInfo("Is Bit Set?", Num.bitIsSet(bitNum))
endMethod
```

bitIsSet example 2

The following example illustrates how you can use **bitIsSet** to display a long integer as a binary number. The **pushButton** method for *showBinary* constructs a string of zeros and ones by testing each bit of a four-byte long integer. For readability, a blank is added to the string every 8 digits.

```
; showBinary::pushButton
method pushButton(var eventInfo Event)
var
  binString  String    ; to construct the binary string
  Num        LongInt
  i          SmallInt  ; for loop index
endVar
if NOT whatNum.isBlank() then
  Num = whatNum                ; get the number test from whatNum
  binString = ""               ; initialize the string
  for i from 0 to 30
    if Num.bitIsSet(LongInt(pow(2, i))) then
      binString = "1" + binString    ; add a 1 to the front of the string
    else
      binString = "0" + binString    ; add a 0 to the front of the string
    endif
    if i = 7 OR i = 15 OR i = 23 then
      binString = " " + binString    ; add a space every 8 digits
    endif
  endfor
  if Num < 0 then
    binString = "1" + binString      ; set the sign bit
  else
    binString = "0" + binString
  endif
  ; show the number
  message("The binary equivalent is ", binString)
endif
endMethod
```

bitOR method

Performs a bitwise OR operation on two values.

Syntax

```
bitOR ( const value LongInt ) LongInt
```

Description

bitOR returns the result of a bitwise OR operation on *value*. **bitOR** operates on the binary representations of two integers and compares them one bit at a time. Here is the truth table for **bitOR**:

a	b	a bitAND b
0	0	0
1	0	1
0	1	1
1	1	1

Example

{button ,AL(`OPAL_TYPE_LONGINT;OPAL_METH_LIBAND;OPAL_METH_LIBXOR;',0,"Defaultoverview",)}
Related Topics

bitOR example

In the following example, the **pushButton** method for a button named *orTwoNums* takes two integers and performs a bitwise OR calculation on them. The result of the calculation is displayed in a dialog box.

```
; orTwoNums::pushButton
method pushButton(var eventInfo Event)
var
  a, b LongInt
endVar
a = 33333  ; binary 00000000 00000000 10000010 00110101
b = -77777 ; binary 11111111 11111110 11010000 00101111
a.bitOR(b) ; binary 11111111 11111110 11010010 00111111
msgInfo("33333 OR -77777", a.bitOR(b)) ; displays -77249
endMethod
```

bitXOR method

Performs a bitwise XOR operation on two values.

Syntax

```
bitXOR ( const value LongInt ) LongInt
```

Description

bitXOR performs a bitwise XOR (exclusive OR) operation on *value*. **bitXOR** operates on the binary representations of two integers and compares them one bit at a time. Here is the truth table for **bitXOR**:

a	b	a bitXOR(b)
0	0	0
1	0	1
0	1	1
1	1	0

Example

{button ,AL(`OPAL_TYPE_LONGINT;OPAL_METH_LIBAND;OPAL_METH_LIBOR;',0,"Defaultoverview",)}

Related Topics

bitXOR example

In the following example, the **pushButton** method for a button named *xorTwoNums* takes two integers and performs a bitwise XOR calculation on them. The result of the calculation is displayed in a dialog box.

```
; xorTwoNums::pushButton
method pushButton(var eventInfo Event)
var
  a, b LongInt
endVar
a = 33333   ; binary 00000000 00000000 10000010 00110101
b = -77777  ; binary 11111111 11111110 11010000 00101111
a.bitXOR(b) ; binary 11111111 11111110 01010010 00011010
msgInfo("33333 XOR -77777", a.bitXOR(b)) ; displays -110054
endMethod
```

LongInt procedure

Casts a value as a LongInt.

Syntax

```
LongInt ( const value AnyType ) LongInt
```

Description

LongInt casts the data type of value to a long integer. If you convert from a more precise type (e.g., Number), precision may be lost.

Example

{button ,AL(`OPAL_TYPE_LONGINT;OPAL_METH_ATVIEW;',0,"Defaultoverview",)} Related Topics

LongInt example

The following example assigns a number to x, casts x to **LongInt**, and assigns the result to l. Notice that the decimal precision of x is lost when it is cast as a **LongInt** and assigned to l.

```
; convertToInt::pushButton
method pushButton(var eventInfo Event)
var
  x Number
  y LongInt
endVar
x = 12.34             ; give x a value
x.view()              ; view x, title of dialog will be "Number"
y = LongInt(x)        ; cast x as a LongInt and assign to y
y.view()              ; show y, note that decimal places are lost
                      ; displays 12 with "LongInt" as title of dialog
endMethod
```

Memo type

Memos contain text and formatting data up to 512MB in Corel Paradox tables. Using Memo type methods **readFromFile** and **writeToFile**, you can transfer memos between forms (and reports), tables, and disk files. You can also use the (=) operator to assign the value of a memo field to a Memo variable or a String variable. Note that there are no arithmetic or comparison operators for Memo variables.
If you assign a memo field to a String variable, you get only the memo text without any formatting. If you assign a memo field to a Memo variable, you get the text and the formatting.

The Memo type includes several derived methods from the AnyType type.

Methods for the Memo type

AnyType	**Memo**
blank	**memo**
dataType	**readFromClipboar**
isAssigned	**readFromFile**
isBlank	**readFromRTFFile**
isFixedType	**writeToClipboard**
	writeToFile
	writeToRTFFile

Print related ObjectPAL methods and examples

memo procedure

Casts a value as a Memo.

Syntax

```
memo ( const value AnyType [ , const value AnyType ] * ) Memo
```

Description

memo casts the expression *value* to a Memo. If you specify multiple arguments, this method will cast all of them to Memos and concatenate them to one Memo.

Example

{button ,AL(`OPAL_TYPE_MEMO;OPAL_METH_MMREADFROMCLIPBOARD;OPAL_METH_MMRFILE;OPAL_METH_MMWRITETOCLIPBOARD;OPAL_METH_MMWFILE;',0,"Defaultoverview",)} **Related Topics**

memo example

The following example assumes that DOCFILES.DB exists and has an alpha field named Memo Name, a Date field named Memo Date, and a formatted memo field named Memo Data. For this example, a form has unbound fields named *stringObject* and *memoObject* and a button named *getMemoData*. The code attached to *getMemoData*'s **pushButton** method defines a TCursor to locate a particular record in *DocFiles*. The code then casts and concatenates the contents of the three *DocFiles* fields to a String value and then to a Memo value. The value cast as a String is displayed in the *stringObject* object and the value cast as a Memo is displayed in the *memoObject* object. When the value is cast as a String, formatting information is not displayed in *stringObject*. When cast as a Memo, *memoObject* displays all formatting information.

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
  tc TCursor
endVar


if tc.open("DocFiles.db") then
  if tc.locate("Memo Name", "Project Notes") then


    ; this line casts data from three DOCFILES.DB fields as a String
    ; because this is cast as a String, the data that appears in stringObject
    ; displays WITHOUT formatting
    stringObject.value = string(tc."Memo Name", "\t",
                                tc."Memo Date", "\n", tc."Memo Data")


    ; this line casts data from three DOCFILES.DB fields as a memo

   ; because this is cast as a MEMO, the data that appears in memoObject
   ; displays with FORMATTED text
   memoObject.value = memo(tc."Memo Name", "\t",
                           tc."Memo Date", "\n", tc."Memo Data")


  else
    msgStop("Error", "Can't find Project Notes.")
  endif
else
  msgStop("Error", "Can't open DocFiles table.")
endif


endMethod
```

readFromClipboard method

Reads text from the Clipboard.

Syntax

readFromClipboard ( ) Logical

Description

readFromClipboard reads text from the Clipboard. **readFromClipboard** will attempt to read in Rich Text Format if the format is available in the Clipboard. Otherwise, text (CF_TEXT) will be read in. This method returns True if successful and False if unsuccessful.

Example

{button ,AL(`OPAL_TYPE_MEMO;OPAL_METH_MMMEMO;OPAL_METH_MMRFILE;OPAL_METH_MMWRITETOCLIPBOARD;OPAL_METH_MMWFILE;',0,"Defaultoverview",)} Related Topics

readFromClipboard example

In the following example, a form has two buttons: *readFromClipboard* and *writeToClipboard*. The first button will read RTF formatted text from the Clipboard into a Memo variable that will then be stored in a table. The second button reads a memo value from a table and writes it to the Clipboard.

The following code is attached to the **pushButton** method for *btnReadFromClipboard*:

```
; btnReadFromClipboard::pushButton
method pushButton(var eventInfo Event)
var
   vrMemo  Memo
   tcMemo  TCursor
endVar


   ;// Open table to hold memos
   tcMemo.open("mymemos.db")
   tcMemo.edit()
   if vrMemo.readFromClipboard() then
      ;// Add a record to the table and insert the value
      tcMemo.insertRecord()
      tcMemo.MemoField = vrMemo
      tcMemo.unlockRecord()
   endIf
   tcMemo.close()


endMethod
```

The following code is attached to the **pushButton** method for *btnWriteToClipboard*:

```
; btnWriteToClipboard::pushButton
method pushButton(var eventInfo Event)
var
   vrMemo  Memo
   tcMemo  TCursor
endVar


   ;// Open table to which contains memos
   tcMemo.open("mymemos.db")
   ;// Make sure there is data in the table
   if tcMemo.nRecords() <> 0 then
      ;// Copy a value to the Memo variable
      vrMemo = tcMemo.MemoField
      ;// Write it out to the Clipboard
      vrMemo.writeToClipboard()
   endIf
   tcMemo.close()


endMethod
```

readFromFile method

Reads a memo from a file.

Syntax

```
readFromFile ( const fileName String ) Logical
```

Description

readFromFile reads a memo from a disk file specified in *fileName*. This method reads text only. It does not read the formatting of formatted memos.

Example

{button ,AL(`OPAL_TYPE_MEMO;OPAL_METH_MMMEMO;OPAL_METH_MMREADFROMCLIPBOARD;OPAL_METH_MMWRITETOCLIPBOARD;OPAL_METH_MMWFILE;',0,"Defaultoverview",)} **Related Topics**

readFromFile example

The following example reads the contents of a text file to a memo field in a table. This examples assumes that a table named *PJNotes* exists in the current directory, and has the following fields: ProjDate, a Date field, and ProjNotes, a Memo field. The **pushButton** method for a button named *getFile* opens, edits, and inserts a new record in the *PJNotes* table, fills the ProjDate field with the current date, and fills the ProjNotes field with text from a file named NOTES.TXT.

```
; getFile::pushButton
method pushButton(var eventInfo Event)
var
  MemoFile Memo
  pTC      TCursor
endVar


if pTC.open("pjNotes.db") then    ; open TCursor for PJNOTES.DB
  if MemoFile.readFromFile("notes.txt") then
    ; if memo file read was successful
    pTC.edit()                    ; edit PJNotes.DB table
    pTC.insertRecord()            ; insert a new blank record
    pTC.ProjDate = today()        ; fill the ProjDate field
    pTC.ProjNotes = MemoFile      ; write memo to ProjNotes field
    pTC.endEdit()                 ; end Edit mode
  endif
  pTC.close()                     ; close the TCursor
endIf
endMethod
```

writeToClipboard method

Writes a memo to the Clipboard.

Syntax

writeToClipboard ( ) Logical

Description

writeToClipboard writes a memo to the Clipboard. The formats copied to the Clipboard are text (CF_TEXT) and Rich Text Format. **writeToClipboard** returns True if successful and False if unsuccessful.

Example

{button ,AL(`OPAL_TYPE_MEMO;OPAL_METH_MMMEMO;OPAL_METH_MMREADFROMCLIPBOARD;OPAL_METH_MMRFILE;OPAL_METH_MMWFILE;',0,"Defaultoverview",)} **Related Topics**

writeToClipboard example

In the following example, a form has two buttons: *readFromClipboard* and *writeToClipboard*. The first button will read Rich Text Format text from the Clipboard into a Memo variable that will then be stored in a table. The second button reads a memo value from a table and writes it out to the Clipboard.

The following code is attached to the **pushButton** method for *btnReadFromClipboard*:

```
; btnReadFromClipboard::pushButton
method pushButton(var eventInfo Event)
var
   vrMemo  Memo
   tcMemo  TCursor
endVar


   ;// Open table to hold memos
   tcMemo.open("mymemos.db")
   if vrMemo.readFromClipboard() then
      ;// Add a record to the table and insert the value
      tcMemo.insertRecord()
      tcMemo.MemoField = vrMemo
      tcMemo.unlockRecord()
   endIf
   tcMemo.close()


endMethod
```

The following code is attached to the **pushButton** method for *btnWriteToClipboard*:

```
; btnWriteToClipboard::pushButton
method pushButton(var eventInfo Event)
var
   vrMemo  Memo
   tcMemo  TCursor
endVar


   ;// Open table to which contains memos
   tcMemo.open("mymemos.db")
   ;// Make sure there is data in the table
   if tcMemo.nRecords() <> 0 then
      ;// Copy a value to the Memo variable
      vrMemo = tcMemo.MemoField
      ;// Write it out to the Clipboard
      vrMemo.writeToClipboard()
   endIf
   tcMemo.close()


endMethod
```

writeToFile method

Writes a memo to a file.

Syntax

```
writeToFile ( const fileName String ) Logical
```

Description

writeToFile writes a memo to a disk file specified in *fileName*. This method writes text only. It does not write the formatting of formatted memos.

Example

{button ,AL(`OPAL_TYPE_MEMO;OPAL_METH_MMMEMO;OPAL_METH_MMREADFROMCLIPBOARD;OPAL_METH_MMRFILE;OPAL_METH_MMWRITETOCLIPBOARD;',0,"Defaultoverview",)} Related Topics

writeToFile example

The following example writes the contents of a memo to a text file. This example assumes that a table named *PJNotes* exists in the current directory, and has the following fields: *ProjDate*, a Date field, and *ProjNotes*, a Memo field. The **pushButton** method for a button named *writeFile* opens the *PJNotes* table, locates a record with the current date, and writes the contents of the *ProjNotes* field for that record to a file named NOTETDAY.TXT.

```
; getFile::pushButton
method pushButton(var eventInfo Event)
var
  MemoFile Memo
  pTC      TCursor
endVar


if pTC.open("pjNotes.db") then             ; open PJNotes.DB table
  if pTC.locate("ProjDate", today()) then
    if NOT (pTC.ProjNotes = blank()) then  ; check if memo is blank
      MemoFile = pTC.ProjNotes             ; if not, write to MemoFile var
      MemoFile.writeToFile("notetday.txt") ; write MemoFile to text file
    endif
  endif
  pTC.close()                              ; close the TCursor
endIf
endMethod
```

writeToRTFFile method

Writes a memo to an RTF file.

Syntax

```
writeToRTFFile ( const fileName String ) Logical
```

Description

writeToRTFFile writes a memo to an RTF disk file specified in *fileName*. This method writes text including the formatting of formatted memos.

Example

{button ,AL(`OPAL_TYPE_MEMO;OPAL_METH_MMMEMO;OPAL_METH_MMREADFROMCLIPBOARD;OPAL_METH_MMRFILE;OPAL_METH_MMWRITETOCLIPBOARD;',0,"Defaultoverview",)} Related Topics

writeToRTFFile example

See the example for **writeToFile**.

readFromRTFFile method

Reads a memo from an RTF file.

Syntax

```
readFromRTFFile ( const fileName String ) Logical
```

Description

readFromRTFFile reads a memo from a disk file specified in *fileName*. This method reads text including the formatting of formatted memos.

Example

{button ,AL(`OPAL_TYPE_MEMO;OPAL_METH_MMMEMO;OPAL_METH_MMREADFROMCLIPBOARD;OPAL_METH_MMWRITETOCLIPBOARD;OPAL_METH_MMWFILE;',0,"Defaultoverview",)} **Related Topics**

readFromRTFFile example

See the example for **readFromFile**.

Menu type

A Menu object is a list of items that appears in the application Menu Bar. When the user chooses an item from a menu, the text of that item is returned. Menus you build in ObjectPAL completely replace Corel Paradox's built-in event menus (but you can get them back using **removeMenu**).

By default, menus do not exist across forms; each form has its own menu system associated with it. If you create a menu for a form, the menu appears only when that form is active. If you then open a second form, the second form uses the built-in event menus, not the menu you created for the first form. If you create a custom menu for each form, you can simulate context-sensitive menus in an application.

If you want two (or more) forms to display the same custom menu, set each form's StandardMenu property to Off. This instructs Corel Paradox to retain the current menu when the user moves from one form to another. You can use the StandardMenu property to construct a single menu system for an entire application.

Note

- A typical application uses both Menu objects and PopUpMenu objects. For more information, see the PopUpMenu type.

Methods for the Menu type

addArray
addBreak
addPopUp
addStaticText
addText
contains
count
empty
getMenuChoiceAttribute
getMenuChoiceAttributeById
hasMenuChoiceAttribute
remove
removeMenu
setMenuChoiceAttribute
setMenuChoiceAttributeById
show

Print related ObjectPAL methods and examples

addArray method

Appends elements of an array to a menu.

Syntax

```
addArray ( const items Array[ ] String )
```

Description

addArray appends *items* from an array to a menu. The array *items* are displayed from left to right across the Menu Bar. To create a drop-down menu or a cascading menu, use **addPopUp**.

Example

{button ,AL(`OPAL_TYPE_MENU;OPAL_METH_MUABRE;OPAL_METH_MUAPOP;OPAL_METH_MUASTA;OPAL_METH_MUATEX;',0,"Defaultoverview",)} Related Topics

addArray example

The following example constructs and displays an application Menu Bar when a form opens. This could be the application's main menu. Throughout the application, the menu displayed here can be changed by methods for other objects.

```
; thisForm::open
method open(var eventInfo Event)
var
  mMenu            Menu     ; main menu
  mmItems Array[3] String   ; main menu items
endVar


if eventInfo.isPreFilter()
  then
    ;code here executes for each object in form
  else
    ;code here executes just for form itself
    ;menu appears when the form first opens
    mmItems[1] = "File"        ; fill the array
    mmItems[2] = "Edit"
    mmItems[3] = "Window"
    mMenu.addArray(mmItems)    ; same as mMenu.addText(...) 3 times
    mMenu.show()               ; show the menu
endif
endMethod
```

addBreak method

Starts a new row in a menu.

Syntax

```
addBreak ( )
```

Description

addBreak starts a new row in a menu. **addBreak** lets you explicitly wrap large menu constructs to two or more rows.

Example

{button ,AL(`OPAL_TYPE_MENU;OPAL_METH_MUAARR;OPAL_METH_MUAPOP;OPAL_METH_MUASTA;OPAL_METH_MUATEX;',0,"Defaultoverview",)} Related Topics

addBreak example

The following example constructs and displays an application Menu Bar when a form opens. It uses **addBreak** to add a second row on the Menu Bar.

```
; thisform::open
method open(var eventInfo Event)
var
  mMenu Menu
endVar
if eventInfo.isPreFilter()
  then
    ;code here executes for each object in form
  else
    ;code here executes just for form itself
    ;menu appears when the form first opens
    mMenu.addText("File")
    mMenu.addText("Edit")
    mMenu.addBreak()
    mMenu.addText("About...")  ; this appears on the second row
    mMenu.show()               ; show the menu
endif
endMethod
```

addPopUp method

Adds a pop-up menu to a Menu Bar item.

Syntax

```
addPopUp ( const menuName String, const cascadedPopup PopUpMenu )
```

Description

addPopUp adds the heading *menuName* and a pop-up menu *cascadedPopup* to a menu. This method is useful for creating drop-down menus and cascading menus.

Note

- If you use **addPopUp** with a *menuName* of &Window, Windows automatically appends a list of open windows to that pop-up menu.

Example

{button ,AL(`OPAL_TYPE_MENU;OPAL_METH_MUAARR;OPAL_METH_MUABRE;',0,"Defaultoverview",)}

Related Topics

addPopUp example

In the following example the code is attached to the built-in **arrive** method for each of two pages of a form. The **arrive** method for *pageOne* creates and displays a custom menu. The **arrive** method for *pageTwo* of the same form removes the custom menu. **addPopUp** is used to create a cascading pop-up menu and a drop-down menu.

Note

- Use SHIFT + F4 to move from the first page to the second. Use SHIFT + F3 to move from the second page to the first.

Here is *pageOne*'s **arrive** method:

```
pageOne::arrive
method arrive(var eventInfo MoveEvent)
var
  p1, p2, p3  PopUpMenu
  m1          Menu
endVar


p1.addText("Passwords...")    ; add items to p1 popup
p1.addText("Attributes...")


p2.addText("Basic...")        ; add items to p2 popup
p2.addText("Scientific...")


p1.addPopUp("Calculator", p2) ; add another item to p1 popup,
                              ; and display p2 popup when the
                              ; item is selected


p3.addText("About...")        ; add an item to 3rd popup


m1.addPopUp("Utilities", p1)  ; add item to Menu Bar,
                              ; and drop-down p1 when selected
m1.addPopUp("Help", p3)       ; add item to Menu Bar,
                              ; and drop-down p3 when selected
m1.show()                     ; show the Menu Bar (not PopUpMenu)


endMethod
```

Here is *pageTwo*'s **arrive** method:

```
; pageTwo::arrive
method arrive(var eventInfo MoveEvent)
  removeMenu()    ; remove the custom menu the default menu
                  ; will appear instead
endMethod
```

addStaticText method

Adds an unselectable text string to a menu.

Syntax

```
addStaticText ( const item String )
```

Description

addStaticText appends *item* to a menu as unselectable text.

Example

{button ,AL(`OPAL_TYPE_MENU;OPAL_METH_MUATEX;',0,"Defaultoverview",)} **Related Topics**

addStaticText example

In the following example, code attached to a form's **open** method creates a Menu Bar. This example uses **addStaticText** to add a static menu item to the Menu Bar:

```
thisForm::open
method open(var eventInfo Event)
var
  mMenu Menu
endVar


if eventInfo.isPreFilter()
  then
    ;code here executes for each object in form
  else
    ;code here executes just for form itself
    mMenu.addStaticText("Main menu")    ; first item is static
    mMenu.addText("File")               ; add two more items
    mMenu.addText("Edit")
    mMenu.show()                        ; show the menu
endif
endMethod
```

addText method

Adds a selectable text string to a menu.

Syntax

```
1. addText ( const menuName String )
2. addText ( const menuName String, const attrib SmallInt )
3. addText ( const menuName String, const attrib SmallInt, const id SmallInt )
```

Description

addText adds a selectable text string to a menu.

Syntax 1 adds the item *menuName* to a menu. Menu items are displayed from left to right across the Menu Bar.

In Syntax 2, you can use *attrib* to preset the display attribute of *menuName*. Use MenuChoiceAttributes constants to specify attributes.

In Syntax 3, you can specify an *id* number (a SmallInt) to identify the menu by number instead of by *menuName*. In the built-in event **menuAction** method, you can use the *id* number to determine which menu the user chooses. When you specify a menu *id*, you should use the built-in IdRanges constant **UserMenu** as a base constant and then add your own number to it or create a user-defined menu constant. In the following example, the code adds File to the *myMenu* menu and specifies an *id* number for that menu item:

```
myMenu.addText("File", MenuEnabled, UserMenu + 1)
```

You can use an ampersand in an item to designate an accelerator key. For example, the item &File would display as File and the user could choose it by pressing ALT + F. If you rely on *menuName* to test for the user's choice, you need to include the ampersand in the comparison string. In the following example, the return value is &File, not File.

To right-align menu items, you can precede *menuName* with the string value \008. After you include \008 in *menuName*, all subsequent menu items appear right-aligned; you don't have to use \008 again. In the following example, the code displays File on the left and Help and Utilities on the right:

```
myMenu.addText("File")
myMenu.addText("\008Help")
myMenu.addText("Utilities")
myMenu.show()
```

Examples

{button ,AL(`OPAL_TYPE_MENU;OPAL_METH_MUASTA;OPAL_METH_MUAARR;',0,"Defaultoverview",)}

Related Topics

addText method examples

Example1 Using the first **addText** syntax

Example2 Using the *id* clause with **addText**

addText example 1

Examples 1 and 2 demonstrate how **addText** syntax influences the way you test for the user's menu choice.

The following example uses the first form of **addText** syntax to create a simple menu. It does not use *id* in the **addText** statements. The code attached to the built-in event **menuAction** method must evaluate the string specified in *menuName* to determine the user's menu choice. The following code is attached to the **open** method for *pageOne*:

```
; pageOne::open
method open(var eventInfo Event)
var
  mainMenu Menu
  utilPU PopUpMenu
endVar


 ; build a pop-up menu
utilPU.addText("&Time")
utilPU.addText("&Date")

 ; attach pop-up to the Utilities main menu item
mainMenu.addPopUp("&Utilities", utilPU)


 ; add "Help" to the menu and right-align "Help" with \008
mainMenu.addText("\008&Help")


 ; now display the menu
mainMenu.show()


endMethod
```

The following code is attached to the **menuAction** method for *pageOne*. This code uses the **menuChoice** method to obtain the string value defined by *menuName*:

```
; pageOne::menuAction
method menuAction(var eventInfo MenuEvent)
var
  choice String
endVar


choice = eventInfo.menuChoice()  ; assign string value to choice


  ; now use choice value to determine which menu was selected
switch
  case choice = "&Time"  :
    msgInfo("Current Time", time())
  case choice = "&Date"  :
    msgInfo("Today's date", today())
  case choice = "\008&Help" :
    ; open the built-in help system
    action(EditHelp)
endSwitch


endMethod
```

addText example 2

The following example demonstrates how you can use the *id* clause with **addText** to refer to menu items by number instead of by name. This code establishes user-defined constants to make it easy to remember the menu *id* assignments. The following code goes in the Const window for *pageOne*:

```
; pageOne::Const
Const
  ; define constants for menu id's
  ; actual values (1, 2 and 3) are arbitrary
  TimeMenu = 1
  DateMenu = 2
  HelpMenu = 3
endConst
```

The following code is attached to the **open** method for *pageOne*. To control the menu display attributes, this code uses built-in constants such as **MenuEnabled**. To identify each menu item by number, the code uses the constants defined in the Const window for *pageOne* (TimeMenu, DateMenu, and HelpMenu).

```
; pageOne::open
method open(var eventInfo Event)
var
  mainMenu Menu
  utilPU PopUpMenu
endVar


 ; build a pop-up menu and use constants (i.e.: TimeMenu)
 ; defined in the Const window for thisPage
utilPU.addText("&Time", MenuEnabled, TimeMenu + UserMenu)
utilPU.addText("&Date", MenuEnabled, DateMenu + UserMenu)

 ; attach pop-up to the Utilities main menu item
mainMenu.addPopUp("&Utilities", utilPU)


 ; add "Help" to the Menu Bar and right-align "Help" with \008
mainMenu.addText("\008&Help", MenuEnabled, HelpMenu + UserMenu)


mainMenu.show()                 ; display the menu


endMethod
```

The following code is attached to the **menuAction** method for *pageOne*. This method evaluates menu selections by *id* number rather than by the name specified in *menuName*.

```
; pageOne::menuAction
method menuAction(var eventInfo MenuEvent)
var
  choice SmallInt
endVar


choice = eventInfo.id()      ; assign constant value (i.e.: 900) to choice


  ; now use constants to determine which menu was selected
switch
  case choice = TimeMenu + UserMenu:
    msgInfo("Current Time", time())
  case choice = DateMenu + UserMenu:
    msgInfo("Today's Date", today())
  case choice = HelpMenu + UserMenu:
    ; open the built-in help system
    action(EditHelp)
endSwitch


endMethod
```

contains method

Reports whether an item is in a menu.

Syntax

```
contains ( const item AnyType ) Logical
```

Description

contains returns True if *item* is in the list of items in a menu; otherwise, it returns False.

Example

{button ,AL(`OPAL_TYPE_MENU;OPAL_METH_MUCOUN;',0,"Defaultoverview",)} **Related Topics**

contains example

The following example assumes that a multi-record object is on the form. When the user changes the value in a field contained in the multi-record object, an Undo menu item is added to the existing custom Menu Bar. When the user moves to another record, Undo is removed. This example uses **contains** to determine if Undo is present before it adds or removes the item. The menu variable is defined in the form's Var window. The Menu Bar is created by the form's **open** method.

The following code goes in the form's Var window:

```
; thisForm::var
Var
  m1 Menu
endVar
```

The following code is for the form's **open** method:

```
; thisForm::open
method open(var eventInfo Event)
if eventInfo.isPreFilter()
  then
    ;code here executes for each object in form
  else
    ;code here executes just for form itself
    m1.addText("&Insert")
    m1.addText("&Delete")
    m1.show()               ; show two item menu
endif
endMethod
```

The following code is for the form's **action** method:

```
; thisForm::action
method action(var eventInfo ActionEvent)
if eventInfo.isPreFilter() then
  ;code here executes for each object in form


  switch
      ; when user locks a record (starts to change a field value)
    case eventInfo.id() = DataLockRecord :
            if not m1.contains("&Undo") then
        ; add Undo and redisplay the menu
        m1.addText("&Undo")
        m1.show()
            endIf


      ; when user posts the record (moves to another record)
    case eventInfo.id() = DataUnlockRecord :
            if m1.contains("&Undo") then
        ; remove Undo redisplay the menu
        m1.remove("&Undo")
        m1.show()
            endIf
  endswitch


endif
endMethod
```

The following code is for the form's **menuAction** method:

```
; thisForm::menuAction
method menuAction(var eventInfo MenuEvent)
var
  choice String
endVar


if eventInfo.isPreFilter() then
  ;code here executes for each object in form


  choice = eventInfo.menuChoice()
```

```
  switch
    case choice = "&Insert" :
      active.action(DataInsertRecord) ; insert new record
    case choice = "&Delete" :
      active.action(DataDeleteRecord) ; delete active record
    case choice = "&Undo"   :
      active.action(DataCancelRecord) ; restore original state
      m1.remove("&Undo")              ; remove Undo menu item
      m1.show()                       ; redisplay menu without Undo
  endswitch

endif
endMethod
```

count method

Returns the number of items in a menu.

Syntax

count ( ) SmallInt

Description

count returns the number of items in a menu, including separators, bars, and breaks.

count returns the number of items in a single menu. If you attach a pop-up menu to a Menu Bar item with **addPopUp**. **count** returns the number of items in the pop-up menu or the number of items in the Menu Bar, but not the total number of items in both menus.

Example

{button ,AL(`OPAL_TYPE_MENU;OPAL_METH_MUCONT;',0,"Defaultoverview",)} **Related Topics**

count example

The following example constructs a menu and a pop-up menu and then displays the number of items in each menu. **count** returns the number of items in a menu whether or not the menu is displayed.

```
; countMenus::pushButton
method pushButton(var eventInfo Event)
var
  m Menu
  p PopUpMenu
endVar


p.addText("&One")
p.addBar()
p.addText("T&wo")
p.addText("Th&ree")       ; 3 items + 1 bar = 4 elements


m.addText("&First")
m.addText("&Second")
m.addPopUp("&Third", p)   ; 3 items in Menu Bar


msgInfo("Menu Bar items", m.count()) ; displays 3 counts Menu Bar only
msgInfo("Pop-up items", p.count())   ; displays 4
counts pop-up only


endMethod
```

empty method

Removes all items from a menu.

Syntax

```
empty ( )
```

Description

empty removes all items from a custom menu. Use **empty** when you need to clear an existing menu before you rebuild it.

Example

{button ,AL(`OPAL_TYPE_MENU;OPAL_METH_MUREMO;',0,"Defaultoverview",)} Related Topics

empty example

The following example uses two buttons to display alternate menus. Both methods affect the same menu, which is declared with the variable *mainMenu* in the form's Var window.

The following code goes in the form's Var window:

```
; thisForm::Var
Var
  mainMenu Menu  ; custom Menu Bar
endVar
```

The following code is for *showMenuOne*'s **pushButton** method:

```
; showMenuOne::pushButton
method pushButton(var eventInfo Event)
  mainMenu.empty()          ; clear the menu
  mainMenu.addText("&One")  ; reconstruct it
  mainMenu.addText("&Two")
  mainMenu.show()           ; display the changed menu
endMethod
```

The following code is for *showMenuTwo*'s **pushButton** method:

```
; showMenuTwo::pushButton
method pushButton(var eventInfo Event)
  mainMenu.empty()              ; clear the menu
  mainMenu.addText("File")      ; reconstruct it
  mainMenu.addText("Edit")
  mainMenu.show()               ; show it again
endMethod
```

getMenuChoiceAttribute procedure

Reports the display attributes of a menu item.

Syntax

```
getMenuChoiceAttribute ( const menuChoice String ) SmallInt
```

Description

getMenuChoiceAttribute returns an integer that represents the display attributes of the menu item specified in *menuChoice*. The integer value represents the combination of attributes that apply. Use MenuChoiceAttributes constants to test attributes. Use **getMenuChoiceAttribute** with **hasMenuChoiceAttribute** to determine whether a specific display attribute applies for a menu item.

This procedure returns the attribute of the currently displayed menu; if you have not created a custom menu, **getMenuChoiceAttribute** operates on the built-in menu.

Example

{button ,AL(`OPAL_TYPE_MENU;OPAL_METH_MUGETMCATTRIBUTEBYID;OPAL_METH_MUHASMENUCHOICEATTRIBUTE;OPAL_METH_MUSETMENUCHOICEATTRIBUTE;OPAL_METH_MUSETMCATTRIBUTEBYID;',0,"Defaultoverview",)} **Related Topics**

getMenuChoiceAttribute example

In the following example, the **open** method for *pageOne* constructs and displays a simple menu. The *getMenuState* button reports whether or not the Time menu item is enabled.

The following code is attached to the **open** method for *pageOne*:

```
; pageOne::open
method open(var eventInfo Event)
var
  mainMenu Menu
  utilPU PopUpMenu
endVar


 ; build a pop-up menu, disable Time option
utilPU.addText("&Time", MenuDisabled + MenuGrayed)
utilPU.addText("&Date")
 ; attach pop-up and show the Menu Bar
mainMenu.addPopUp("&Utilities", utilPU)
mainMenu.addText("&Help")
mainMenu.show()


endMethod
```

The following code is for *getMenuState*'*s* **pushButton** method:

```
; getMenuState::pushButton
method pushButton(var eventInfo Event)
var
  attrib SmallInt
endVar


  ; store attributes of Time in attrib
attrib = getMenuChoiceAttribute("&Time")
  ; this displays False because Time is disabled
msgInfo("Time enabled?", HasMenuChoiceAttribute(attrib, MenuEnabled))
  ; this displays True because Time is grayed
msgInfo("Time grayed?", hasMenuChoiceAttribute(attrib, MenuGrayed))


endMethod
```

getMenuChoiceAttributeById procedure

Reports the display attribute of a menu item specified by its menu ID.

Syntax

```
getMenuChoiceAttributeById ( const menuId SmallInt ) SmallInt
```

Description

getMenuChoiceAttributeById returns an integer that represents the display attributes of the menu item specified in *menuId*. The integer value represents the combination of attributes that apply. Use MenuChoiceAttributes constants to test attributes. Use **getMenuChoiceAttributeById** with **hasMenuChoiceAttribute** to determine whether a specific display attribute applies for a menu item.

This procedure returns the attribute of the currently displayed menu; if you have not created a custom menu, **getMenuChoiceAttributeById** operates on the built-in menu.

This procedure is similar to **getMenuChoiceAttribute** in that both report the display attributes for a specified menu item. The difference is that you specify the actual menu ID (a SmallInt value) for **getMenuChoiceAttributeById** and the menu name (a String value) for **getMenuChoiceAttribute**. **getMenuChoiceAttributeById** is especially useful when you specify a menu ID as part of **addText** syntax.

 **Example**

{button ,AL(`OPAL_TYPE_MENU;OPAL_METH_MUGMCA;OPAL_METH_MUHASMENUCHOICEATTRIBUTE;OPAL_METH_MUSETMENUCHOICEATTRIBUTE;',0,"Defaultoverview",)} Related Topics

getMenuChoiceAttributeById example

The following example demonstrates how you can use **getMenuChoiceAttributeById** with **hasMenuChoiceAttribute** to determine whether a menu item is disabled. In this example, the **open** method for *pageOne* constructs a small menu. The **pushButton** method for the *getMenuState* button reports on the state of the Undo menu item.

The following code goes in the form's Var window:

```
; thisForm::Var
Var
  m1     Menu
  p1, p2 PopUpMenu
endVar
```

The following code goes in the form's Const window:

```
; thisForm::Const
Const
  UndoMenu  = 1
  InsMenu   = 2
  DelMenu   = 3
  IndexMenu = 4
  AboutMenu = 5
endConst
```

The following code is for the page's **open** method:

```
; pageOne::open
method open(var eventInfo Event)


p1.addText("Undo",   MenuDisabled + MenuGrayed, UndoMenu + UserMenu)
p1.addText("Insert", MenuEnabled, InsMenu + UserMenu)
p1.addText("Delete", MenuEnabled, DelMenu + UserMenu)
p2.addText("Index",  MenuEnabled, IndexMenu + UserMenu)
p2.addText("About",  MenuEnabled, AboutMenu + UserMenu)


m1.addPopUp("&Record", p1)
m1.addPopUp("&Help", p2)
m1.show()


endMethod
```

The following code is attached to the *getMenuState*'*s* **pushButton** method:

```
; getMenuState::pushButton
method pushButton(var eventInfo Event)


  ; store attributes of Undo menu in attrib
attrib = getMenuChoiceAttributeById(UndoMenu + UserMenu)


  ; this displays False because Undo is disabled
msgInfo("Undo enabled?", hasMenuChoiceAttribute(attrib, MenuEnabled))
  ; this displays True because Undo is grayed
msgInfo("Undo grayed?", hasMenuChoiceAttribute(attrib, MenuGrayed))
endMethod
```

hasMenuChoiceAttribute procedure

Reports whether a menu item contains a given display attribute.

Syntax

```
hasMenuChoiceAttribute ( const attrib SmallInt , const attribSet SmallInt ) Logical
```

Description

hasMenuChoiceAttribute returns True if *attribSet* contains the attribute specified in *attrib*; otherwise, it returns False. Use MenuChoiceAttributes constants to specify attributes.

Use **hasMenuChoiceAttribute** with **getMenuChoiceAttribute** or **getMenuChoiceAttributeById** to determine whether a particular display attribute for a menu item is represented in *attribSet*.

Example

{button ,AL(`OPAL_TYPE_MENU;OPAL_METH_MUGMCA;OPAL_METH_MUGETMCATTRIBUTEBYID;',0,"Defaultoverview",)} **Related Topics**

hasMenuChoiceAttribute example

The following example demonstrates how you can use **hasMenuChoiceAttribute** with **getMenuChoiceAttribute** to determine whether a particular attribute applies to the currently displayed menu.

The following code is attached to the **open** method for *pageOne*:

```
; pageOne::open
method open(var eventInfo Event)
var
  m1 Menu
  p1 PopUpMenu
endVar


p1.addText("&Insert")   ; create a simple menu
p1.addText("&Delete")
p1.addText("&Undo")
m1.addPopUp("&Record", p1)
m1.show()


endMethod
```

The following code is attached to the **pushButton** method for the *toggleMenuState* button:

```
; toggleMenuState::pushButton
method pushButton(var eventInfo Event)
var
  attribSet SmallInt
endVar


 ; store composite menu attributes in attribSet
attribSet = getMenuChoiceAttribute("&Undo")


 ; this is True if Undo is enabled
if hasMenuChoiceAttribute(attribSet, MenuEnabled) then
   setMenuChoiceAttribute("&Undo", MenuDisabled + MenuGrayed)
else
  setMenuChoiceAttribute("&Undo", MenuEnabled)
endif


endMethod
```

remove method

Removes an item from a menu.

Syntax

```
remove ( const item AnyType )
```

Description

remove deletes the first occurrence of *item* from a menu. This method is used to change one item in a menu without having to rebuild the entire menu.

Example

{button ,AL(`OPAL_TYPE_MENU;OPAL_METH_MUCONT;OPAL_METH_MUEMPT;',0,"Defaultoverview",)}

Related Topics

remove example

The following example changes a menu immediately by removing an item and adding another item in its place.

```
; changeMenu::pushButton
method pushButton(var eventInfo Event)
var
   mainMenu Menu
endVar


; First, assume the user is working with a form.
; You could display a menu like this:
mainMenu.addText("File")
mainMenu.addText("Edit")
mainMenu.addText("Form")
mainMenu.show()
msgInfo("Status", "About to change menus. Watch closely.")


; Then, suppose the user switches to work on a report.
; You could change the menu like this:
mainMenu.remove("Form")
mainMenu.addText("Report")
mainMenu.show()


msgInfo("Status", "About to remove the menus. Watch closely.")


; remove entire menu, reveal built-in menus
removeMenu()
endMethod
```

removeMenu procedure

Removes a custom menu and displays the default menu.

Syntax

```
removeMenu ( )
```

Description

removeMenu replaces a menu built using ObjectPAL with Corel Paradox's default menu.

Example

{button ,AL(`OPAL_TYPE_MENU;OPAL_METH_MUEMPT;OPAL_METH_MUREMO;',0,"Defaultoverview",)}

Related Topics

removeMenu example

In the following example, the form's **open** method constructs a menu (but does not display it). The arrive method for *pageOne* displays the menu with **show**. The **arrive** method for *pageTwo* removes the menu and reveals the built-in Corel Paradox menu.

The following code goes in the form's Var window:

```
; thisForm::var
Var
  m1 Menu
endVar
```

The following code is attached to the form's **open** method:

```
; thisForm::open
method open(var eventInfo Event)
if eventInfo.isPreFilter()
  then
    ;code here executes for each object in form
  else
    ;code here executes just for form itself


     m1.addText("&File")   ; construct a menu
     m1.addText("&Edit")
     m1.addText("For&m")


endif


endMethod
```

The following code is attached to the **arrive** method for *pageOne*:

```
; pageOne::arrive
method arrive(var eventInfo MoveEvent)
m1.show()  ; display the application menu
endMethod
```

The following code is attached to the **arrive** method for *pageTwo*:

```
; pageTwo::arrive
method arrive(var eventInfo MoveEvent)
removeMenu()   ; remove application menu, reveal built-in menu
endMethod
```

setMenuChoiceAttribute procedure

Sets the display attribute of a menu item.

Syntax

```
setMenuChoiceAttribute ( const menuChoice String, const menuAttribute SmallInt )
```

Description

setMenuChoiceAttribute sets the display attribute of *menuChoice* to *menuAttribute*. Use MenuChoiceAttributes constants to specify attributes. This procedure affects the currently displayed menu; if you have not created a custom menu, **setMenuChoiceAttribute** affects the built-in menu.

Note

- If a menu item's definition includes an accelerator key (for example, Print which is defined as &Print), remember to include the ampersand in the comparison string *menuChoice*.

Example

{button ,AL(`OPAL_TYPE_MENU;OPAL_METH_MUGMCA;OPAL_METH_MUGETMCATTRIBUTEBYID;OPAL_METH_MUHASMENUCHOICEATTRIBUTE;OPAL_METH_MUSETMCATTRIBUTEBYID;',0,"Defaultoverview",)}

Related Topics

setMenuChoiceAttribute example

The following example changes the attribute of the Undo option, depending on whether there is anything to undo. As the user makes changes to the record, the Undo item can be selected. After posting the changes, Undo is unavailable.

The following code goes in the form's Var window:

```
; thisForm::var
Var
  m1 Menu
  p1 PopUpMenu
endVar
```

The following code is for the form's **open** method:

```
; thisForm::open
method open(var eventInfo Event)
if eventInfo.isPreFilter()
  then
    ;code here executes for each object in form
  else
    ;code here executes just for form itself


    ; create a menu and show it
    p1.addText("&Undo", MenuDisabled + MenuGrayed)
    p1.addText("&Insert")
    p1.addText("&Delete")
    m1.addPopUp("&Record", p1)
    m1.show()


endif


endMethod
```

The following code is for the form's **action** method:

```
; thisForm::action
method action(var eventInfo ActionEvent)


if eventInfo.isPreFilter()
  then
    ;code here executes for each object in form


    switch
         ; when user locks a record (starts to change a field value)
      case eventInfo.id() = DataLockRecord :
         ; enable Undo menu item
        setMenuChoiceAttribute("&Undo", MenuEnabled)


        ; when user posts the record (moves to another record)
      case eventInfo.id() = DataUnlockRecord :
        ; disable and gray Undo menu item
        setMenuChoiceAttribute("&Undo", MenuDisabled + MenuGrayed)
    endswitch


  else
    ;code here executes just for form itself
  endif


endMethod
```

The following code is for the form's **menuAction** method:

```
; thisForm::menuAction
method menuAction(var eventInfo MenuEvent)
var
  choice String
endVar
```

```
if eventInfo.isPreFilter()
  then
    ;code here executes for each object in form

    choice = eventInfo.menuChoice()
    switch
      case choice = "&Insert" :
        active.action(DataInsertRecord)  ; insert new record
      case choice = "&Delete" :
        active.action(DataDeleteRecord)  ; delete active record
      case choice = "&Undo"   :
        active.action(DataCancelRecord)  ; revert record to original state
        setMenuChoiceAttribute("&Undo", MenuDisabled + MenuGrayed)
    endswitch


  else
    ;code here executes just for form itself
endif

endMethod
```

setMenuChoiceAttributeById procedure

Sets the display attribute of a menu item.

Syntax

```
setMenuChoiceAttributeById ( const menuId String, const menuAttribute SmallInt )
```

Description

setMenuChoiceAttributeById sets the display attribute of *menuId* to *menuAttribute*. Use MenuChoiceAttributes constants to specify attributes. This procedure affects the currently displayed menu; if you have not created a custom menu, **setMenuChoiceAttributeById** affects the built-in menu.

Note

- If a menu item's definition includes an accelerator key (e.g., Print which is defined as &Print), remember to include the ampersand in the comparison string *menuChoice*.

Example

{button ,AL(`OPAL_TYPE_MENU;OPAL_METH_MUGMCA;OPAL_METH_MUGETMCATTRIBUTEBYID;OPAL_METH_MUHASMENUCHOICEATTRIBUTE;OPAL_METH_MUSETMENUCHOICEATTRIBUTE;',0,"Defaultoverview",)}

Related Topics

setMenuChoiceAttributeById example

The following example changes the attribute of the Undo option, depending on whether there is anything to undo. As the user makes changes to the record, the Undo item can be selected. After posting the changes, Undo is unavailable. This example uses the *menuId* clause in **addText** so that the code can refer to menu items by number rather than by menu name.

The following code goes in the form's Var window:

```
; thisForm::var
Var
  m1 Menu
  p1 PopUpMenu
endVar
```

The following code goes in the form's Const Window:

```
; thisForm::const
Const
  InsMenu  = 1  ; use constants for menu id's
  DelMenu  = 2
  UndoMenu = 3
endConst
```

The following code is attached to the form's **open** method:

```
; thisForm::open
method open(var eventInfo Event)


if eventInfo.isPreFilter()
  then
    ;code here executes for each object in form
  else
    ;code here executes just for form itself


     ; construct a menu and display it
     p1.addText("&Undo",   MenuDisabled + MenuGrayed, UndoMenu + UserMenu)
     p1.addText("&Delete", MenuEnabled, DelMenu + UserMenu)
     p1.addText("&Insert", MenuEnabled, InsMenu + UserMenu)
     m1.addPopUp("&Record", p1)
     m1.show()


  endif


endMethod
```

The following code is attached to the form's **action** method:

```
; thisForm::action
method action(var eventInfo ActionEvent)


if eventInfo.isPreFilter()
  then
    ;code here executes for each object in form


    switch
         ; when user locks a record (starts to change a field value)
      case eventInfo.id() = DataLockRecord :
         ; enable Undo menu item
        setMenuChoiceAttributeById(UndoMenu + UserMenu,
                                   MenuEnabled)


         ; when user posts the record (moves to another record)
      case eventInfo.id() = DataUnlockRecord :
         ; disable and dim Undo menu item
        setMenuChoiceAttributeById(UndoMenu + UserMenu,
                                   MenuGrayed + MenuDisabled)
    endswitch


  else
```

```
    ;code here executes just for form itself
endif

endMethod
```

The following code is attached to the form's **menuAction** method:

```
; thisForm::menuAction
method menuAction(var eventInfo MenuEvent)
var
  menuItem SmallInt
endVar

if eventInfo.isPreFilter() then
  ;code here executes for each object in form

  menuItem = eventInfo.id()
  switch
    case menuItem = InsMenu :
      active.action(DataInsertRecord)    ; insert new record
    case menuItem = DelMenu :
      active.action(DataDeleteRecord)    ; delete active record
    case menuItem = UndoMenu :
      active.action(DataCancelRecord)    ; revert record to original state
      setMenuChoiceAttributeById(UndoMenu, MenuDisabled + MenuGrayed)
  endswitch

  endswitch

else
  ;code here executes just for form itself
endif

endMethod
```

show method

Displays a menu.

Syntax

```
show ( )
```

Description

show displays a menu.

The user's choice is handled using the built-in event methods **menuAction** and **menuChoice** from the MenuEvent type.

Example

{button ,AL(`OPAL_TYPE_MENU;OPAL_METH_MUATEX;OPAL_METH_FOSETMENU;',0,"Defaultoverview",)}

Related Topics

show example

In the following example, a form's **open** method constructs a simple menu and displays it with **show**. The **menuAction** method for the form handles the user's menu choice. The following code is attached to the **open** method for *thisForm*.

```
; thisForm::open
method open(var eventInfo Event)
var
  p1 PopUpMenu
  m1 Menu
endVar


if eventInfo.isPreFilter()
  then
    ;code here executes for each object in form
  else
    ;code here executes just for form itself


    p1.addText("&Time")           ; construct a pop-up
    p1.addText("&Date")
    m1.addPopUp("&Utilities", p1) ; attach pop-up to menu item
    m1.show()                     ; display the m1 menu

endif


endMethod
```

The following code is attached to the form's **menuAction** method:

```
; thisForm::menuAction
method menuAction(var eventInfo MenuEvent)
var
  menuName String
endVar


if eventInfo.isPreFilter() then
  ;code here executes for each object in form


  menuName = eventInfo.menuChoice()
  switch
    case menuName = "&Time" : msgInfo("Current Time", time())
    case menuName = "&Date" : msgInfo("Today's Date", date())
  endSwitch


else
  ;code here executes just for form itself
endif


endMethod
```

MenuEvent type

MenuEvent variables contain data related to menu selections in the application Menu Bar. When the user chooses an item from a menu, it triggers the built-in **menuAction** method. By modifying an object's built-in **menuAction** method, you can define how the object responds.

The MenuEvent type includes several derived methods from the Event type.

Methods for the MenuEvent type

Event	**MenuEvent**
errorCode	**data**
getTarget	**id**
isFirstTime	**isFromUI**
isPreFilter	**menuChoice**
isTargetSelf	**reason**
setErrorCode	**setData**
	setId
	setReason

{button ,AL(`MENUEV;',0,"Defaultoverview",)} **Related Topics**

Print related ObjectPAL methods and examples

User-defined menu constants

You can define your own menu constants, but you must keep them within a specific range. Because this range is subject to change in future versions of Paradox, ObjectPAL provides the IdRanges constants UserMenu and UserMenuMax to represent the minimum and maximum values allowed.

The following example supposes that you want to define two menu constants, ThisMenuItem and ThatMenuItem. You would define values for your custom constants in a Const window as follows:

```
Const
   ThisMenuItem = 1
   ThatMenuItem = 2
EndConst
```

To use one of these constants, you would add it to UserMenu. For example,

```
method menuAction(var eventInfo MenuEvent)
   if eventInfo.id() = UserMenu + ThisMenuItem then
      doSomething()
   endIf
endMethod
```

By adding UserMenu to your own constant, you guarantee yourself a value above the minimum. To keep the value under the maximum, use the value of UserMenuMax. One way to check the value is with the following **message** statement:

```
message(UserMenuMax)
```

In this version of Paradox, the difference between UserMenu and UserMenuMax is 2047. That means the largest value you can use for a menu constant is UserMenu + 2047.

{button ,AL(`MENUEV;',0,"Defaultoverview",)} Related Topics

data method

Returns information about a MenuEvent.

Syntax

data ( ) LongInt

Description

data should be used by Windows programmers only. **data** returns the *lParam* argument (usually zero) of specific Windows messages, such as WM_SYSCOMMAND and WM_COMMAND. For more information, see your Windows programming documentation.

{button ,AL(`OPAL_METH_MNID;OPAL_METH_MNSETDATA;OPAL_METH_MNSID;',0,"Defaultoverview",) } Related Topics

id method

Returns the ID of a MenuEvent.

Syntax

```
id ( ) SmallInt
```

Description

id returns the ID number of a MenuEvent. ObjectPAL provides MenuCommands constants (like MenuFileOpen) for many common menu choices. You can also use user-defined menu constants to test the value returned by **id**.

Examples

{button ,AL(`OPAL_METH_MNSID;',0,"Defaultoverview",)} Related Topics

id method examples

Example1	Using a form's built-in **menuAction** method
Example2	Using a menu ID argument with **addText**
Example3	Using the ID MenuCanClose

id example 1

The following example attaches code to a form's built-in **menuAction** method. When the user selects Close from the System menu, attempts to toggle to a design window, or chooses File, Exit, the method asks the user to confirm whether or not to leave the form.

```
; thisForm::menuAction
method menuAction(var eventInfo MenuEvent)
if eventInfo.isPreFilter()
  then
    ; code here executes for each object in form
  else
    ; code here executes just for form itself
    if eventInfo.id() = MenuControlClose OR
      eventInfo.id() = MenuFileExit OR
      eventInfo.id() = MenuFormDesign then
      disableDefault                      ; block departure
      ans = msgQuestion("Please confirm",
                        "Do you really want to leave?")
      if ans = "Yes" then
        dodefault
      endif
    endif
endif
endMethod
```

id example 2

The following example demonstrates how you can use the menu ID argument with **addText** to refer to menu items by number (ideally, user-defined constants) instead of by name. This code establishes user-defined constants to make it easy to remember the menu ID assignments.

The following code defines constants global to *pageOne*:

```
; pageOne::Const
Const
  ; define constants for menu IDs
  ; actual values (1, 2 and 3) are arbitrary
  TimeMenu = 1
  DateMenu = 2
  HelpMenu = 3
endConst
```

The following code is attached to the **open** method for *pageOne*. To control the menu display attributes, this code uses built-in constants such as MenuEnabled. To identify each menu item by number, the code uses the constants defined in the Const window for *pageOne* (TimeMenu, DateMenu, and HelpMenu).

```
; pageOne::open
method open(var eventInfo Event)
var
  mainMenu Menu
  utilPU PopUpMenu
endVar


 ; build a pop-up menu and use constants (i.e.: TimeMenu)
 ; defined in the Const window for thisPage
utilPU.addText("&Time", MenuEnabled, TimeMenu + UserMenu)
utilPU.addText("&Date", MenuEnabled, DateMenu + UserMenu)
 ; UserMenu is an ObjectPAL constant
 ; attach pop-up to the Utilities main menu item
mainMenu.addPopUp("&Utilities", utilPU)


 ; add "Help" to the Menu Bar and right-justify "Help" with \008
mainMenu.addText("\008&Help", MenuEnabled, HelpMenu) + UserMenu


mainMenu.show()                 ; display the menu


endMethod
```

The following code is attached to the **menuAction** method for *pageOne*. This method evaluates menu selections by ID number rather than by the name specified in *menuName*.

```
; pageOne::menuAction
method menuAction(var eventInfo MenuEvent)
var
  choice SmallInt
endVar


choice = eventInfo.id()      ; assign constant value to choice


  ; now use constants to determine which menu was selected
switch
  case choice = TimeMenu + UserMenu:
    msgInfo("Current Time", time())
  case choice = DateMenu + UserMenu:
    msgInfo("Today's Date", today())
  case choice = HelpMenu + UserMenu:
    ; change menu ID to built-in constant (MenuHelpContents)
    ; this effectively opens the built-in help system.
    eventInfo.setId(MenuHelpContents)
    eventInfo.setReason(MenuDesktop)
endSwitch


endMethod
```

id example 3

The following example shows you how to use the menu action event and to test for the ID MenuCanClose. This will display a message before the user can close the form. To stop the closure of the form, use setErrorCode and provide any non zero value.

```
method MenuAction (var eventInfo MenuEvent)

if eventInfo.isPrefilter() then

else
   if eventInfo.id() = MenuCanClose then
      if msgQuestion("Exit?", "Are you sure?") = "No" then
         eventInfo.setErrorCode(1)   ;// Any non-zero error code works
      endif
   endif
endif

endMethod
```

isFromUI method

Reports whether an event was generated by the user interacting with Corel Paradox.

Syntax

isFromUI ( ) Logical

Description

isFromUI reports whether an event was generated by the user interacting with Corel Paradox, or internally (e.g., by an ObjectPAL statement). This method returns True if the event was generated by the user; otherwise, it returns False.

Examples

{button ,AL(`OPAL_METH_EVISPREFILTER;',0,"Defaultoverview",)} Related Topics

isFormUI method examples

Example1	Checking for a Delete menu action
Example2	Indicating the source of a menu action

isFromUI example 1

The following example checks for a menu action to delete a record. If the action is from the UI (that is, if the user made the menu choice), a dialog box prompts for confirmation before the record is deleted.

```
;frm :: menuAction
method menuAction(var eventInfo MenuEvent)


   if eventInfo.isPreFilter() then
      ;// This code executes for each object on the form:

   else
      ;// This code executes only for the form:


      if eventInfo.id() = MenuRecordDelete and
      eventInfo.isFromUI() then
            if msgQuestion("Delete record?",
                     "Delete this record?") <> "Yes" then


               disableDefault
               return
            endIf
         endIf
   endif


endMethod
```

isFromUI example 2

The following example shows how you can use **isFromUI** to indicate if the menu action was sent by **menuAction** or by **sendKeys**.

The following code is attached to the page's Const window. It declares constants to make it easy to remember the menu ID assignments.

```
; pageOne::Const
Const
  ; define constants for menu IDs
  ; actual values (1, 2 and 3) are arbitrary
  kTimeMenu = 1
  kDateMenu = 2
  kHelpMenu = 3
endConst
```

The following code is attached to the **open** method for *pageOne*. To control the menu display attributes, this code uses ObjectPAL constants such as MenuEnabled. To identify each menu item by number, the code uses the constants defined in the Const window for *pageOne* (kTimeMenu, kDateMenu, and kHelpMenu).

```
; pageOne::open
method open(var eventInfo Event)
var
  mainMenu Menu
  utilPU    PopUpMenu
endVar


 ; build a pop-up menu and use constants (i.e.: kTimeMenu)
 ; defined in the Const window for thisPage
utilPU.addText("&Time", MenuEnabled, kTimeMenu + UserMenu)
utilPU.addText("&Date", MenuEnabled, kDateMenu + UserMenu)
 ; UserMenu is an ObjectPAL constant
 ; attach pop-up to the Utilities main menu item
mainMenu.addPopUp("&Utilities", utilPU)


 ; add "Help" to the Menu Bar and right-justify "Help" with \008
mainMenu.addText("\008&Help", MenuEnabled, kHelpMenu + UserMenu)


mainMenu.show()                ; display the menu


endMethod
```

The following code is attached to the **menuAction** method for *pageOne*. This method evaluates menu selections by ID number rather than by the name specified in *menuName*. In addition, it uses **isFromUI** to report whether the menu event was generated by **menuAction** or by **keyPhysical**.

```
; pageOne::menuAction
method menuAction(var eventInfo MenuEvent)
var
  choice          SmallInt
  youDoneIt   Logical
endVar


youDoneIt = eventInfo.isFromUI()
choice = eventInfo.id()      ; assign constant value to choice


  ; now use constants to determine which menu was selected
switch
  case choice = kTimeMenu + UserMenu:
    msgInfo("Did a user do this", youDoneIt)
    msgInfo("Current Time", time())
  case choice = kDateMenu + UserMenu:
    msgInfo("Did a user do this", youDoneIt)
    msgInfo("Today's Date", today())
  case choice = kHelpMenu + UserMenu:
    ; change menu ID to built-in constant (MenuHelpContents)
    ; this effectively opens the built-in help system.
    eventInfo.setId(MenuHelpContents)
    eventInfo.setReason(MenuDesktop)
```

```
endSwitch
endMethod
```

The following two buttons demonstrate the use of the code above. The following code is attached to the **pushButton** method of a button named *btnObjectPAL*. It uses **menuAction** to send a menu event:

```
;btnObjectPAL :: pushButton
method pushButton(var eventInfo Event)
   menuAction(kDateMenu + UserMenu)
endMethod
```

The following code is attached to the **pushButton** method of a button named *btnSendKeys*. It uses **sendKeys** to send the keystrokes ALT + u + t . Use this button to simulate a user selecting a menu.

```
;btnSendKeys :: pushButton
method pushButton(var eventInfo Event)
   sendKeys("%ut")
endMethod
```

menuChoice method

Returns a string that contains an item chosen from a menu.

Syntax

```
menuChoice ( ) String
```

Description

menuChoice returns a string that contains an item chosen from a menu. Use **menuChoice** to modify an object's built-in **menuAction** method to specify how that object responds to menu choices.

If the definition of a menu item includes an accelerator key (e.g., &Print), remember to include the ampersand in the comparison string. The following example compares the return value of **menuChoice** with the string &Print:

```
if eventInfo.menuChoice() = "&Print" then
   ; print the report
endif
```

Example

{button ,AL(`OPAL_METH_MNID;OPAL_METH_MNSID;',0,"Defaultoverview",)} Related Topics

menuChoice example

The following example assumes a form contains at least one memo field, named *thisMemoField*. When the user arrives on *thisMemoField*, the built-in **arrive** method displays a menu that lets the user perform basic cut and paste operations. The built-in **menuAction** method attached to *thisMemoField* uses **menuChoice** to evaluate the user's selection and to take appropriate action. Although this example mimics the behavior of the default menus, this technique is necessary when the default menus are replaced by custom menus.

The following code is attached to the built-in **arrive** method for *thisMemoField*:

```
; thisMemoField::arrive
method arrive(var eventInfo MoveEvent)
Var
  EditPopUp PopUpMenu
  EditMenu  Menu
endVar


EditPopUp.addText("&Cut")              ; create a pop-up menu
EditPopUp.addText("&Copy")
EditPopUp.addText("&Paste")


EditMenu.addPopUp("&Edit", EditPopUp)  ; add pop-up Menu Bar item
EditMenu.show()                        ; display the menu
endMethod
```

The following code is attached to the built-in **menuAction** method for *thisMemoField*. Note that comparisons in the **switch...endSwitch** statement must include the ampersand, such as &Cut:

```
thisMemoField::menuAction
method menuAction(var eventInfo MenuEvent)
var
  choice String
endVar
choice = eventInfo.menuChoice()       ; store the menu selection to choice


; now respond to the selection appropriately
switch
  case choice = "&Cut"   : self.action(EditCutSelection)
  case choice = "&Copy"  : self.action(EditCopySelection)
  case choice = "&Paste" : self.action(EditPaste)
endSwitch
endMethod
```

The following code is attached to the built-in **depart** method for *thisMemoField*. When the user leaves *thisMemoField*, this code removes the menu. In this example, the default menus reappear when the user moves off the field. In a similar situation, you might want to display another custom menu structure.

```
; thisMemoField::depart
method depart(var eventInfo MoveEvent)
removeMenu()          ; remove the Edit menu
endMethod
```

reason method

Reports the type of menu chosen.

Syntax

```
reason ( ) SmallInt
```

Description

reason returns an integer value to report why a MenuEvent occurred. MenuEvent reasons occur when a built-in **menuAction** method is called. ObjectPAL provides MenuReasons constants to test the value returned by **reason**.

Example

{button ,AL(`OPAL_METH_MNSREA;',0,"Defaultoverview",)} Related Topics

reason example

In the following example, the form's **menuAction** method examines every MenuEvent to determine the reason for the MenuEvent. The reason is then displayed in the *menuReasonField* field object.

```
; thisForm::menuAction
method menuAction(var eventInfo MenuEvent)
var
  reasonStr String
endVar
  if eventInfo.isPreFilter() then
    ; sort out the reason, and assign equivalent string to reasonStr
    reasonStr = iif(eventInfo.reason() = MenuNormal, "MenuNormal",
                iif(eventInfo.reason() = MenuControl, "MenuControl",
                 "MenuDesktop"))
    reasonId = eventInfo.reason()
    menuReasonField = String(reasonId) + " " + reasonStr
     ; Code here executes before each object
  else
     ; Code here executes afterwards (or for form)

  endif
endMethod
```

setData method

Specifies information about a MenuEvent.

Syntax

```
setData ( const menuData LongInt )
```

Description

setData should be used by Windows programmers only. **setData** specifies the *lParam* argument (usually zero) of specific Windows messages, such as WM_SYSCOMMAND and WM_COMMAND. For more information, see your Windows programming documentation.

{button ,AL(`OPAL_METH_MNDATA;OPAL_METH_MNID;OPAL_METH_MNSID;',0,"Defaultoverview",)}
Related Topics

setId method

Specifies the ID of a MenuEvent.

Syntax

```
setId ( const commandId SmallInt )
```

Description

setId specifies in *commandId* an action to take as the result of a menu choice, where *commandId* is a MenuCommands constant.

If you change the ID for a MenuEvent with **setId**, you may also need to change the reason for that MenuEvent with **setReason**.

In many circumstances, you should use **menuAction** from the Form type or UIObject type to invoke a menu command. Although it is possible to change the reason and ID for an existing MenuEvent (*eventInfo*), and it is also possible to create a new MenuEvent and set the reason and ID for that event (only advanced users should try this), this technique is not always advisable.

Example

{button ,AL(`OPAL_METH_MNID;OPAL_METH_EVERRORCODE;OPAL_METH_EVSETERRORCODE;OPAL_METH_FOMENUACTION;OPAL_METH_UIMENUACTION;',0,"Defaultoverview",)} Related Topics

setId example

See the **id** example.

setReason method

Specifies a reason for generating a MenuEvent.

Syntax

```
setReason ( const reasonId SmallInt )
```

Description

setReason specifies in *reasonId* a reason for generating a MenuEvent, where *reasonId* is a MenuReasons constant.

In many circumstances, you should use **menuAction** from the Form type or UIObject type to invoke a menu command. Although it is possible to change the reason and ID for an existing MenuEvent (*eventInfo*), and it is also possible to create a new MenuEvent and set the reason and ID for that event (only advanced users should try this), this technique is not always advisable.

Example

{button ,AL(`OPAL_METH_MNREAS;OPAL_METH_MNID;OPAL_METH_FOMENUACTION;OPAL_METH_UIMENUACTION;',0,"Defaultoverview",)} Related Topics

setReason example

See the **id** example.

MouseEvent type

A MouseEvent object answers questions about the mouse, including

- where the mouse is located
- was a mouse button clicked
- which mouse button was clicked or held down during an operation

The following built-in object variables are useful when you work with the MouseEvents lastMouseClicked and lastMouseRightClicked.

Many methods defined for the MouseEvent type use or return Point values. Methods defined for the Point type get and set information about screen coordinates and relative positions of points. For example, the size and position properties of a design object are specified in points.

ObjectPAL calculates point values relative to the container of the design object in question. For example, if a box contains a button, ObjectPAL calculates the button's position relative to the box. If the button sits in an empty page, ObjectPAL calculates the button's position relative to the page. Methods that take or return Point values as arguments use this relative framework. The method **convertPointWithRespectTo** defined for the UIObject type can be used to convert values in different frameworks.

The following built-in event methods are triggered by MouseEvents: **mouseClick**, **mouseDown**, **mouseUp**, **mouseDouble**, **mouseRightUp**, **mouseRightDown**, **mouseRightDouble**, **mouseMove**, **mouseEnter**, and **mouseExit**.

The following table displays the methods for the MouseEvent type, including several derived methods from the Event type.

Methods for the MouseEvent type

Event	**MouseEvent**
errorCode	**getMousePosition**
getTarget	**getObjectHit**
isFirstTime	**isControlKeyDown**
isPreFilter	**isFromUI**
isTargetSelf	**isInside**
reason	**isLeftDown**
setErrorCod	**isMiddleDown**
setReason	**isRightDown**
	isShiftKeyDown
	setControlKeyDown
	setInside
	setLeftDown
	setMiddleDown
	setMousePosition
	setRightDown
	setShiftKeyDown
	setX
	setY
	x
	y

Print related ObjectPAL methods and examples

getMousePosition method

Returns the mouse position as a Point.

Syntax

```
1. getMousePosition ( var p Point )
2. getMousePosition ( var xPosition LongInt, yPosition LongInt )
```

Description

getMousePosition returns the mouse position. This method gets the mouse position at the time the method was called. It doesn't track subsequent mouse movements.

Syntax 1 stores the value in a Point variable, *p*. When you use Syntax 1, you can use Point type methods (for example, **isLeft** and **isRight**) to get more information.

Syntax 2 stores the value in *xPosition* and *yPosition*, two LongInt variables the represent the x and y coordinates of the pointer.

Example

{button ,AL(`OPAL_TYPE_MOUSEEVENT;OPAL_METH_MESPOS;OPAL_METH_MEGETOBJECTHIT;OPAL_METH_EVGTAR;OPAL_METH_SYGMOUS;OPAL_METH_SYSMPOS;',0,"Defaultoverview",)} **Related Topics**

getMousePosition example

The following example gets the position of the last **mouseUp** event and draws a small circle at that position. The method checks if the source of the event was from the UI (in this case, from the user) and if the target of the event is the page itself (as opposed to whether it was bubbled up to the page from some other object). This method draws the circle only when the user clicks on the page:

```
; pageOne::mouseUp
method mouseUp(var eventInfo MouseEvent)
var
  crObj UIObject
  x, y  LongInt    ; point coordinates
endVar
if eventInfo.isFromUI() AND eventInfo.isTargetSelf() then
  ; create a small blue circle at the mouse position
  eventInfo.getMousePosition(x, y)
  crObj.create(ellipseTool, x, y, 1440, 1440)
  crObj.Color = DarkBlue
  crObj.Visible = True
endif
endMethod
```

getObjectHit method

Creates a handle to the UIObject that received the event.

Syntax

```
getObjectHit ( var target UIObject ) Logical
```

Description

getObjectHit returns in *target* a handle to the UIObject that was clicked. This method is useful for the internal MouseEvents that call the built-in event methods **mouseExit** and **mouseEnter**. **getObjectHit** can return a different object than **getTarget** during a **mouseExit** or **mouseEnter** method.

Example

{button ,AL(`OPAL_TYPE_MOUSEEVENT;OPAL_METH_MEGPOS;OPAL_METH_EVGTAR;',0,"Defaultoverview",)} Related Topics

getObjectHit example

The following method is attached to the **mouseExit** method of a form. When the mouse exits an object, a message appears in the Status Window showing the name of the target object (**getTarget**) and the name of the object hit (**getObjectHit**).

```
; thisForm::mouseExit
method mouseExit(var eventInfo MouseEvent)
var
  targObj,
  hitObj   UIObject
endVar
if eventInfo.isPreFilter()
  then
    ;code here executes for each object in form
    eventInfo.getTarget(targObj)
    eventInfo.getObjectHit(hitObj)
    message(targObj.Name + " vs. " + hitObj.Name)
  else
    ;code here executes just for form itself

endif
endMethod
```

isControlKeyDown method

Reports whether the user has held (or is holding) down CTRL during a MouseEvent.

Syntax

```
isControlKeyDown ( ) Logical
```

Description

isControlKeyDown returns True if CTRL is held down during a MouseEvent; otherwise, it returns False.

Example

{button ,AL(`OPAL_TYPE_MOUSEEVENT;OPAL_METH_MESCKD;OPAL_METH_MEISKD;',0,"Defaultoverview",)} Related Topics

isControlKeyDown example

The following example examines the keyboard state during a mouse click to determine whether to automatically insert the highest value in the range, the lowest value in a range, or the default value.

The following constants are declared in the Const window for *fieldOne*:

```
; fieldOne::Const
Const
  HighRangeVal = Number(10000)
  LowRangeVal = Number(100000)
  DefaultVal = Number(50000)
endConst
```

The following code is the method for **mouseUp** for *fieldOne*:

```
; fieldOne::mouseUp
method mouseUp(var eventInfo MouseEvent)
; insert high, low, or default value depending on how mouse was clicked
switch
  case eventInfo.isControlKeyDown() : self.Value = LowRangeVal
                                      message("CTRL-click")
  case eventInfo.isShiftKeyDown()   : self.Value = HighRangeVal
                                      message("SHIFT-click")
  otherwise                         : self.Value = LowRangeVal
                                      message("Click")
endswitch
endMethod
```

isFromUI method

Reports whether an event was generated by the user interacting with Corel Paradox.

Syntax

isFromUI ( ) Logical

Description

isFromUI reports whether an event was generated by the user interacting with Corel Paradox, or internally (e.g., by an ObjectPAL statement). This method returns True if the event was generated by the user; otherwise, it returns False.

Example

{button ,AL(`OPAL_TYPE_MOUSEEVENT;OPAL_METH_EVISPREFILTER;',0,"Defaultoverview",)} Related Topics

isFromUI example

Sometimes you need to know whether a MouseEvent was generated by the user interacting with the form or by ObjectPAL; for example, in a computer tutorial. In the following example, **isFromUI** is used to determine whether a button's built-in **mouseEnter** method was triggered by the user or by ObjectPAL:

```
;btnOpenCust :: mouseEnter
method mouseEnter(var eventInfo MouseEvent)
   if eventInfo.isFromUI() then
      message("This button opens the customer form.")
   else
      message("After you press this button, the customer form opens.")
   endIf
endMethod
```

isInside method

Reports whether the mouse is inside the border of the target object.

Syntax

isInside ( ) Logical

Description

isInside reports whether the mouse is within the border of the target object at the time of the event.

Example

{button ,AL(`OPAL_TYPE_MOUSEEVENT;OPAL_METH_MEGETOBJECTHIT;OPAL_METH_MESEIN;',0,"Default overview",)} Related Topics

isInside example

In the following example, the **mouseUp** method for *buttonOne* reports whether the last event is inside the borders of the target object. If you click *buttonOne*, the **mouseUp** MouseEvent is delivered to *buttonOne* and **isInside** returns True. If you drag from inside the button to outside the button, so that the **mouseUp** occurs outside of the borders of *buttonOne*. The MouseEvent occurs for *buttonOne*, and triggers the **mouseUp** method, but **isInside** returns False for that MouseEvent.

```
; buttonOne::mouseUp
method mouseUp(var eventInfo MouseEvent)
msgInfo("Is the last event inside ?", eventInfo.isInside())
endMethod
```

isLeftDown method

Reports whether the left mouse button is held down during a MouseEvent.

Syntax

```
isLeftDown ( ) Logical
```

Description

isLeftDown returns True if the left mouse button is held down during a MouseEvent, for example, while dragging the mouse; otherwise, it returns False.

Example

{button ,AL(`OPAL_TYPE_MOUSEEVENT;OPAL_METH_MEIRDN;OPAL_METH_MEIMDN;OPAL_METH_MESLDN;',0,"Defaultoverview",)} Related Topics

isLeftDown example

In the following example, assume that the *Site Notes* field from the *Sites* table is placed on a form. This method, attached to the **mouseMove** method for *Site Notes*, checks whether the left or right mouse button is down at the time of the move. If the left mouse button is down, the field is selected from the point of the click to the beginning of the field. If the right mouse button is down, the field is selected from the point of the click to the end of the field.

```
; Site Notes::mouseMove
method mouseMove(var eventInfo MouseEvent)
if eventInfo.isLeftDown() then
  self.action(SelectTop)            ; select from point to beginning
else
  if eventInfo.isRightDown() then
    self.action(SelectBottom)       ; select from point to end
  endif
endif
endMethod
```

isMiddleDown method

Reports whether the middle mouse button is held down during a MouseEvent.

Syntax

isMiddleDown ( ) Logical

Description

isMiddleDown returns True if the middle mouse button is held down during a MouseEvent; otherwise (even if there is no middle mouse button), it returns False.

Example

{button ,AL(`OPAL_TYPE_MOUSEEVENT;OPAL_METH_MESMDN;OPAL_METH_MEILDN;OPAL_METH_MEIRDN;',0,"Defaultoverview",)} Related Topics

isMiddleDown example

The following example assumes that a form contains a button called *sendMove* and a field from the *Sites* table called *Site Notes*. The **pushButton** method for *sendMove* constructs a MouseEvent with the middle button down and then sends the MouseEvent off to the *Site Notes* field.

```
; sendMove::pushButton
method pushButton(var eventInfo Event)
var
  mo  MouseEvent         ; declare a MouseEvent to send
  ui  UIObject
endVar
ui.attach("Site Notes")  ; attach to Site Notes
mo.setMiddleDown(Yes)    ; set middle button down on MouseEvent
ui.mouseMove(mo)         ; dispatch event to mouseMove for Site Notes
endMethod
```

The following method is attached to the **mouseMove** method for *Site Notes*. If the middle button is down for the MouseEvent, the method moves to the beginning of the current word and then selects the entire word.

```
; Site_Notes::mouseMove
method mouseMove(var eventInfo MouseEvent)
if eventInfo.isMiddleDown() then
  self.action(MoveLeftWord)    ; go to the beginning of the word
  self.action(SelectRightWord) ; select the entire word
endif
endMethod
```

isRightDown method

Reports whether the right mouse button is held down during a MouseEvent.

Syntax

isRightDown ( ) Logical

Description

isRightDown returns True if the right mouse button is held down during a MouseEvent, for example, while right-dragging; otherwise, it returns False.

Example

{button ,AL(`OPAL_TYPE_MOUSEEVENT;OPAL_METH_MESRDN;OPAL_METH_MEILDN;OPAL_METH_MEIMDN;',0,"Defaultoverview",)} Related Topics

isRightDown example

In the following example, assume that the *Site Notes* field from the *Sites* table is placed on a form. The **mouseMove** method for *Site Notes* checks whether the left or right mouse button is down at the time of the move. If the left mouse button is down, the field is selected from the point of the click to the beginning of the field; if the right mouse button is down, the field is selected from the point of the click to the end of the field.

```
; Site Notes::mouseMove
method mouseMove(var eventInfo MouseEvent)
if eventInfo.isLeftDown() then
  self.action(SelectTop)            ; select from point to beginning
else
  if eventInfo.isRightDown() then
    self.action(SelectBottom)       ; select from point to end
  endif
endif
endMethod
```

isShiftKeyDown method

Reports whether SHIFT is held down during a MouseEvent.

Syntax

```
isShiftKeyDown ( ) Logical
```

Description

isShiftKeyDown returns True if SHIFT is held down during a MouseEvent; otherwise, it returns False.

Example

{button ,AL(`OPAL_TYPE_MOUSEEVENT;OPAL_METH_MEICKD;OPAL_METH_MESSKD;',0,"Defaultoverview",)} Related Topics

isShiftKeyDown example

The following example is attached to the **mouseUp** method for the *Site Notes* field. When the user presses SHIFT while clicking, the word to the right of the cursor is selected.

```
; Site Notes::mouseUp
method mouseUp(var eventInfo MouseEvent)
;if SHIFT is down, select the word to the right
if eventInfo.isShiftKeyDown() then
   self.action(SelectRightWord)
endif
endMethod
```

setControlKeyDown method

Simulates pressing and holding CTRL during a MouseEvent.

Syntax

```
setControlKeyDown ( const yesNo Logical )
```

Description

setControlKeyDown adds information about the state of CTRL for a MouseEvent. You must specify Yes or No. Yes means CTRL was pressed and held during a MouseEvent; No means CTRL was not pressed.

Example

{button ,AL(`OPAL_TYPE_MOUSEEVENT;OPAL_METH_MEICKD;OPAL_METH_MESSKD;',0,"Defaultoverview",)} Related Topics

setControlKeyDown example

The following example creates a MouseEvent and sets CTRL to Yes. The event is then sent to the **mouseUp** built-in event method for a field called *lcField*. This method is attached to the **pushButton** method for a button named *sendCTRL*:

```
; sendCTRL::pushButton
method pushButton(var eventInfo Event)
var
  CTRLMsEvent MouseEvent             ; declare the event
endVar


CTRLMsEvent.setControlKeyDown(Yes)   ; set the Control key
lcField.mouseUp(CTRLMsEvent)         ; send the event
endMethod
```

The following code is attached to the **mouseUp** method for *lcField*. This method checks whether CTRL is pressed when the mouse is clicked. If so, the value in the field is changed to all lowercase.

```
; lcField::mouseUp
method mouseUp(var eventInfo MouseEvent)
if eventInfo.isControlKeyDown() then   ; check for Control key
  self.Value = lower(self.Value)       ; change to lowercase
endif
endMethod
```

setInside method

Sets the mouse to be inside the current object.

Syntax

```
setInside ( const TrueFalse Logical ) Logical
```

Description

setInside sets the MouseEvent to be inside the current object.

Example

{button ,AL(`OPAL_TYPE_MOUSEEVENT;OPAL_METH_MEISINSIDE;OPAL_METH_MEGETOBJECTHIT;',0,"Defaultoverview",)} **Related Topics**

setInside example

In the following example, the **mouseUp** method for *sendAnEvent* uses **setInside** to change the *eventInfo* variable and then sends the event to *buttonOne*.

```
; sendAnEvent::mouseUp
method mouseUp(var eventInfo MouseEvent)
eventInfo.setInside(Yes)
buttonOne.mouseUp(eventInfo)
endMethod
```

setLeftDown method

Simulates clicking the left mouse button.

Syntax

```
setLeftDown ( const yesNo Logical )
```

Description

setLeftDown adds information about the state of the left mouse button for a MouseEvent. You must specify Yes or No. Yes means the left mouse button was clicked; No means the left mouse button was not clicked.

Example

{button ,AL(`OPAL_TYPE_MOUSEEVENT;OPAL_METH_MEILDN;OPAL_METH_MESMDN;OPAL_METH_MESRDN;',0,"Defaultoverview",)} Related Topics

setLeftDown example

The following example constructs a MouseEvent with the left mouse button set down. The MouseEvent is then sent to the **mouseMove** method for *Site_Notes*. The following code is attached to the **pushButton** method for *sendLeftButton*:

```
; sendLeftButton::pushButton
method pushButton(var eventInfo Event)
var
  leftMoveMouse MouseEvent     ; create the mouse event
  ui            UIObject
endVar
leftMoveMouse.setLeftDown(Yes) ; set Left button to Yes
ui.attach("Site_Notes")
ui.mouseMove(leftMoveMouse)    ; send the event to Site_Notes
endMethod
```

The following code is attached to the **mouseMove** method for *Site Notes*:

```
; Site_Notes::mouseMove
method mouseMove(var eventInfo MouseEvent)
if eventInfo.isLeftDown() then
  self.action(SelectTop)            ; select from point to beginning
else
  if eventInfo.isRightDown() then
    self.action(SelectBottom)       ; select from point to end
  endif
endif
endMethod
```

setMiddleDown method

Simulates clicking the middle mouse button.

Syntax

```
setMiddleDown ( const yesNo Logical )
```

Description

setMiddleDown adds information about the state of the middle mouse button for a MouseEvent. You must specify Yes or No. Yes means the middle button was clicked; No means the middle mouse button was not clicked.

Example

{button ,AL(`OPAL_TYPE_MOUSEEVENT;OPAL_METH_MEIMDN;OPAL_METH_MESLDN;OPAL_METH_MESRDN;',0,"Defaultoverview",)} Related Topics

setMiddleDown example

The following example assumes that a form contains a button called *sendMove* and a field object from the *Sites* table called *Site_Notes*. The **pushButton** method for *sendMove* constructs a MouseEvent with the middle mouse button down and then sends MouseEvent to the *Site_Notes* field object.

```
; sendMove::pushButton
method pushButton(var eventInfo Event)
var
  mo  MouseEvent         ; declare a MouseEvent to send
  ui  UIObject
endVar
ui.attach("Site_Notes")  ; attach to Site_Notes
mo.setMiddleDown(Yes)    ; set middle button down on MouseEvent
ui.mouseMove(mo)         ; dispatch event to mouseMove for Site Notes
endMethod
```

The following method is attached to the **mouseMove** method for *Site_Notes*. If the middle button is down for the MouseEvent, the method moves to the beginning of the current word and then selects the entire word.

```
; Site_Notes::mouseMove
method mouseMove(var eventInfo MouseEvent)
if eventInfo.isMiddleDown() then
  self.action(MoveLeftWord)    ; go to the beginning of the word
  self.action(SelectRightWord) ; select the entire word
endif
endMethod
```

setMousePosition method

Sets the position of the mouse for an event.

Syntax

```
1. setMousePosition ( const xPosition LongInt, const yPosition  LongInt )
2. setMousePosition ( const p Point )
```

Description

setMousePosition adds information about the position of the mouse for a MouseEvent. *xPosition* and *yPosition* specify the x and y coordinates in twips, relative to the upper-left corner of the target object's container.

Example

{button ,AL(`OPAL_TYPE_MOUSEEVENT;OPAL_METH_MEGPOS;OPAL_METH_SYGMOUS;OPAL_METH_SYSMPOS;',0,"Defaultoverview",)} Related Topics

setMousePosition example

The following example creates a new event, sets the mouse position to 500 twips to the right and below the current mouse position, and sends the event to the **mouseRightUp** method for the same object.

The following code is attached to the **mouseUp** method for an object called *boxOne*:

```
; boxOne::mouseUp
method mouseUp(var eventInfo MouseEvent)
var
  rightEvent MouseEvent
endVar
; set the new position to current plus 500, 500
rightEvent.setMousePosition(eventInfo.x() + 500,
                            eventInfo.y() + 500)
mouseRightUp(rightEvent)            ; send off the new event
endMethod
```

setRightDown method

Simulates clicking the right mouse button.

Syntax

```
setRightDown ( const yesNo Logical )
```

Description

setRightDown adds information about the state of the right mouse button for a MouseEvent. You must specify Yes or No. Yes means the right mouse button was clicked; No means the right mouse button was not clicked.

Example

{button ,AL(`OPAL_TYPE_MOUSEEVENT;OPAL_METH_MEIRDN;OPAL_METH_MESMDN;OPAL_METH_MESLDN;',0,"Defaultoverview",)} Related Topics

setRightDown example

The following example constructs a MouseEvent with the right mouse button set down. The MouseEvent is then sent to the **mouseMove** method for *Site_Notes*. This code is attached to the **pushButton** method for *sendRightButton*:

```
; sendRightButton::pushButton
method pushButton(var eventInfo Event)
var
  rightMoveMouse MouseEvent      ; declare the event
  ui             UIObject
endVar
rightMoveMouse.setRightDown(Yes) ; set right button down
ui.attach("Site_Notes")
ui.mouseMove(rightMoveMouse)     ; send the event to Site Notes
endMethod
```

The following code is attached to the **mouseMove** method for *Site_Notes*:

```
; Site_Notes::mouseMove
method mouseMove(var eventInfo MouseEvent)
if eventInfo.isLeftDown() then
  self.action(SelectTop)            ; select from point to beginning
else
  if eventInfo.isRightDown() then
    self.action(SelectBottom)       ; select from point to end
  endif
endif
endMethod
```

setShiftKeyDown method

Simulates pressing and holding SHIFT.

Syntax

```
setShiftKeyDown ( const yesNo Logical )
```

Description

setShiftDown adds information about the state of SHIFT for a MouseEvent. You must specify Yes or No. Yes means SHIFT was pressed and held; No means SHIFT was not pressed.

Example

{button ,AL(`OPAL_TYPE_MOUSEEVENT;OPAL_METH_MEISKD;OPAL_METH_MESCKD;',0,"Defaultoverview",)} Related Topics

setShiftKeyDown example

The following example creates a MouseEvent and sets SHIFT to Yes. The event is then sent to the **mouseUp** built-in event method for a field called *ucField*. This method is attached to the **pushButton** method for a button named *sendShift*

```
; sendShift::pushButton
method pushButton(var eventInfo Event)
var
  ShiftMsEvent MouseEvent             ; declare the event
endVar


ShiftMsEvent.setShiftKeyDown(Yes)     ; set the SHIFT key
ucField.mouseUp(ShiftMsEvent)         ; send the event

endMethod
```

The following code is attached to the **mouseUp** method for *ucField*. This method checks whether SHIFT is pressed when the mouse is clicked. If so, the value in the field is changed to all uppercase.

```
; ucField::mouseUp
method mouseUp(var eventInfo MouseEvent)
if eventInfo.isShiftKeyDown() then    ; check for SHIFT key
  self.Value = upper(self.Value)      ; change to uppercase
endif
endMethod
```

setX method

Specifies the horizontal coordinate of the mouse-pointer position.

Syntax

```
setX ( const xPosition LongInt )
```

Description

setX sets the horizontal coordinate (in twips) of the mouse-pointer position to *xPosition*. Coordinates must be specified relative to the upper-left corner of the current object.

Example

{button ,AL(`OPAL_TYPE_MOUSEEVENT;OPAL_METH_MEX;OPAL_METH_MEY;OPAL_METH_MESETY;OPAL_METH_UICONV;',0,"Defaultoverview",)} Related Topics

setX example

The following example involves two methods for the same object, *boxOne*. The **mouseUp** method creates a MouseEvent, setting the coordinates to 500 twips greater than the point of the click. The **mouseUp** method then sends the event to **mouseRightUp**. The **mouseRightUp** method gets the coordinates, converts them so they are placed properly on *boxOne*, and draws a box at the point indicated by the MouseEvent. If the MouseEvent is the result of a user interaction (**isFromUI** returns True), the new box is painted Red. If the MouseEvent is not the result of a user interaction, like when the event is passed from the **mouseUp** method, the new box is painted Green. The **mouseUp** method for *boxOne* is:

```
; boxOne::mouseUp
method mouseUp(var eventInfo MouseEvent)
var
  rightEvent MouseEvent
endVar
; set the new position to current plus 500, 500
rightEvent.setX(eventInfo.x() + 500)
rightEvent.setY(eventInfo.y() + 500)
mouseRightUp(rightEvent)            ; send off the new event
endMethod
```

The following code is attached to the **mouseRightUp** method for *boxOne*:

```
; boxOne::mouseRightUp
method mouseRightUp(var eventInfo MouseEvent)
var
  ui     UIObject     ; to create object at point of click
  msPt   Point        ; the x, y point of click
endVar


; get the x and y coordinates of the click
msPt = Point(eventInfo.x(), eventInfo.y())


; convert the point from the page to the box
self.convertPointWithRespectTo(pageOne, msPt, msPt)


; create the box, color it, and set it to visible
ui.create(boxTool, msPt.x(), msPt.y(), 200, 200)
ui.Visible = True
if eventInfo.isFromUI() then
  ui.Color = Red         ; native event
else
  ui.Color = Green       ; mouse event passed from mouseUp
endif
endMethod
```

setY method

Specifies the vertical coordinate of the mouse-pointer position.

Syntax

```
setY ( const yPosition LongInt )
```

Description

setY sets the vertical coordinate (in twips) of the mouse-pointer position to *yPosition*. Coordinates must be specified relative to the upper-left corner of the current object.

Example

{button ,AL(`OPAL_TYPE_MOUSEEVENT;OPAL_METH_MEX;OPAL_METH_MEY;OPAL_METH_MESETX;OPAL_METH_UICONV;',0,"Defaultoverview",)} Related Topics

setY example

See the **setX** example.

x method

Returns the horizontal coordinate of the mouse-pointer position.

Syntax

```
x ( ) LongInt
```

Description

x returns (in twips) the horizontal coordinate of the mouse-pointer position.

Example

{button ,AL(`OPAL_TYPE_MOUSEEVENT;OPAL_METH_MEY;OPAL_METH_MESETX;OPAL_METH_UICONV;',0,"Defaultoverview",)} Related Topics

x example

See the **setX** example.

y method

Returns the vertical coordinate of the mouse-pointer position.

Syntax

```
y ( ) LongInt
```

Description

y returns (in twips) the vertical coordinate of the mouse-pointer position.

Example

{button ,AL(`OPAL_TYPE_MOUSEEVENT;OPAL_METH_MEX;OPAL_METH_MESETY;OPAL_METH_MESETX;OPAL_METH_UICONV;',0,"Defaultoverview",)} Related Topics

y example

See the **setX** example.

MoveEvent type

Methods for the MoveEvent type enable you to get and set information about the events that occur as you navigate from one object to another in a form.

The following built-in event methods are triggered by MoveEvents: **arrive**, **canArrive**, **canDepart**, and **depart**.

The MoveEvent type includes several derived methods from the Event type.

Methods for the MoveEvent type

Event	**MoveEvent**
errorCode	**getDestination**
getTarget	**reason**
isFirstTime	**setReason**
isPreFilter	
isTargetSelf	
setErrorCode	

Print related ObjectPAL methods and examples

getDestination method

Reports which object is the destination of a move.

Syntax

```
getDestination ( var dest UIObject )
```

Description

getDestination returns in *dest* the object that Corel Paradox is trying to move to in a form.

Example

{button ,AL(`OPAL_TYPE_MOVEEVENT;OPAL_METH_MVREASON;OPAL_METH_EVGTAR;',0,"Defaultoverview",)} **Related Topics**

getDestination example

In the following example, assume that the form contains a multi-record object bound to the *Orders* table. The **canDepart** method for the form is called whenever the user attempts to move off a field or other object in the form. The **canDepart** method shown in this example uses **getDestination** to find the intended destination of the MoveEvent. This method uses **getTarget** to find the source of the move and compare it with the destination.

If the containers of the two objects are the same, such as when the user is moving from one field to the next in a multi-record object, the method displays a dialog box asking for confirmation. When the user responds, the move occurs and the field the user moved from is set to yellow. If the target's container and the destination's container are different, such as when the user is attempting to leave the form altogether, the method doesn't display the dialog box.

The following code is attached to the **canDepart** method for a form:

```
; thisForm::canDepart
method canDepart(var eventInfo MoveEvent)
var
  destObj UIObject
  targObj UIObject
  doMove  String
endVar
if eventInfo.isPreFilter()
  then
    ;code here executes for each object in form
    eventInfo.getTarget(targObj)
    eventInfo.getDestination(destObj)
    if targObj.ContainerName = destObj.ContainerName then
      ; handle only field-to-field moves within the MRO
      doMove = msgQuestion("Move?", "Move to " + destObj.name + " ?")
      if doMove = "No" then
        eventInfo.setErrorCode(CanNotDepart)
      else
        targObj.Color = Yellow    ; leave a trail of yellow fields
      endIf
    endIf
  else
    ;code here executes just for form itself

endIf
endMethod
```

reason method

Reports why a move occurred.

Syntax

```
reason ( ) SmallInt
```

Description

reason returns an integer value to report why a MoveEvent occurred. MoveEvent reasons occur when a built-in **arrive**, **depart**, **canArrive**, or **canDepart** method is called. ObjectPAL provides MoveReasons constants for testing the value returned by **reason**.

Example

{button ,AL(`OPAL_TYPE_MOVEEVENT;OPAL_METH_MVSETREASON;',0,"Defaultoverview",)} Related Topics

reason example

In the following example, assume a form contains two field objects, *fieldOne* and *fieldTwo*, and a button named *moveToFieldOne*. A move away from *fieldOne* is treated as normal; however, to return to *fieldOne*, the user must press the *moveToFieldOne* button. The **canArrive** method for *fieldOne* checks the reason for the move and blocks field arrival if the reason is not UserMove.

The following code is attached to the **canArrive** method for *fieldOne*:

```
; fieldOne::canArrive
method canArrive(var eventInfo MoveEvent)
; don't allow user to move to field by tabbing or clicking
if eventInfo.reason() = UserMove then
  eventInfo.setErrorCode(CanNotArrive)
  beep()
  message("Press the Move to Field One button to move to Field One.")
endIf
endMethod
```

The following code is attached to the **pushButton** method for *moveToFieldOne*:

```
; moveToFieldOne::pushButton
method pushButton(var eventInfo Event)
; move to fieldOne if it does not currently have focus
if fieldOne.Focus = False then
  fieldOne.moveTo()
else
  fieldTwo.moveTo()
endIf
endMethod
```

setReason method

Specifies a reason for a Move Event.

Syntax

```
setReason ( const reasonId SmallInt )
```

Description

setReason specifies a reason for generating a MoveEvent. This method takes a MoveReasons constant as an argument.

Example

{button ,AL(`OPAL_TYPE_MOVEEVENT;OPAL_METH_MVREASON;',0,"Defaultoverview",)} Related Topics

setReason example

In the following example, the **canArrive** method for *fieldOne* blocks field arrival if the reason for the move is UserMove. To temporarily circumvent this restriction, the form's **canArrive** method changes the reason for UserMove events to PalMove events.

The following code is attached to the **canArrive** method for *fieldOne*:

```
; fieldOne::canArrive
method canArrive(var eventInfo MoveEvent)
; don't allow user to move to field by tabbing or clicking
if eventInfo.reason() = UserMove then
  eventInfo.setErrorCode(CanNotArrive)
  beep()
  message("Press the Move to Field One button to move to Field One.")
endIf
endMethod
```

The following code is attached to the **canArrive** method for the form:

```
; thisForm::canArrive
method canArrive(var eventInfo MoveEvent)
if eventInfo.isPreFilter()
  then
    ;code here executes for each object in form
    ; change events with a reason of UserMove to PalMove
    if eventInfo.reason() = UserMove then
      eventInfo.setReason(PalMove)
    endIf
  else
    ;code here executes just for form itself

endIf
endMethod
```

Number type

Number variables represent floating-point values consisting of a significand (fractional portion, for example, 3.224) multiplied by a power of 10. The significand contains up to 18 significant digits, and the power of 10 ranges from ± 3.4E-4930 to ± 1.1E4930. Assigning values outside of this range to a Number variable causes an error.

The following code demonstrates ObjectPAL's alternate syntax:

methodName (objVar, argument [, argument])

methodName is the name of the method, *objVar* is the variable representing an object, and *argument* represents one or more arguments. For example, the following statement uses the standard ObjectPAL syntax to return the sine of a number:

```
theNum.sin()
```

The following statement uses the alternate syntax:

```
sin(theNum)
```

Use ObjectPAL's standard syntax for clarity and consistency and use the alternate syntax only where convenient.

Although the numeric method's display formats may vary depending on the user's Windows number format, ObjectPAL's internal representation is always the same.

Run-time library methods and procedures defined for the Number type also work with LongInt and SmallInt variables. In all cases, the syntax is the same, and the returned value is a Number. Although sin does not appear in the list of methods for the LongInt type, the following code executes:

```
var
   abc LongInt
   xyz Number
endVar
abc = 43
xyz = abc.sin()
```

The following table displays the methods of the Number type, including several derived methods from the AnyType type:

Methods for the Number type

AnyType	**Number**
blank	**abs**
dataType	**acos**
isAssigned	**asin**
isBlank	**atan**
isFixedType	**atan2**
view	**ceil**
	cos
	cosh
	exp
	floor
	fraction
	fv
	ln
	log
	max
	min
	mod
	number
	numVal
	pmt
	pow
	pow10
	pv

rand
round
sin
sinh
sqrt
tan
tanh
truncate

Print related ObjectPAL methods and examples

abs method

Returns the absolute value of a number.

Syntax

abs ( ) Number

Description

abs removes the sign from a number.

Example

{button ,AL(`OPAL_TYPE_NUMBER;OPAL_METH_NUNUMB;',0,"Defaultoverview",)} **Related Topics**

abs example

The following example assumes that a form contains three field objects: *forecastAmt*, *actualAmt*, and *diffPercent*. The **newValue** method for *actualAmt* calculates the difference between *forecastAmt* and *actualAmt* and then determines the accuracy of the forecast. The difference between *forecastAmt* and *actualAmt* can be positive or negative. **abs** returns the absolute value of the number, which is then multiplied by 100 to determine the percentage of error. This code is attached to the **newValue** method for *actualAmt*::

```
; actualAmt::newValue
method newValue(var eventInfo Event)
var
  difference  Number
endVar
; don't execute if newValue is being called at startup, or
; if one of the fields involved is blank
if eventInfo.reason() <> StartupValue then
  if NOT self.isBlank() AND
     NOT forecastAmt.isBlank() then
    ; find out how much forecast differs from actual
    difference = (forecastAmt - Number(self.Value)) / forecastAmt
    diffPercent = difference.abs() * 100  ; get the variation as
                                          ; an absolute value
  else
    msgStop("Error", "The forecastAmt field can't be blank.")
  endIf
endIf
endMethod
```

acos method

Returns the 2-quadrant arc cosine of a number.

Syntax

acos ( ) Number

Description

Given a number between -1 and 1, **acos** returns a numeric value between 0 and pi, expressed in radians. **acos** is called the 2-quadrant arc cosine because it returns values within quadrants 1 and 4 (i.e., between -pi/2 and pi/2). **acos** is the inverse of **cos** if **acos**(*x*) = *y* and then **cos**(*y*) = *x*.

Example

{button ,AL(`OPAL_TYPE_NUMBER;OPAL_METH_NUASIN;OPAL_METH_NUATAN;',0,"Defaultoverview",)}

Related Topics

acos example

The following example uses **pushButton** method for the *findArcCos* button to calculate and display the arc cosine of a value:

```
; findArcCos::pushButton
method pushButton(var eventInfo Event)
   var
      nuUserVal,
      nuArcCos   Number
      stPrompt   String
   endVar


   stPrompt = "Enter a number from -1 to 1"
   nuUserVal = 0


   nuUserVal.view(stPrompt)
   if (nuUserVal >= -1) and (nuUserVal <= 1) then
      nuArcCos = nuUserVal.acos()
      nuArcCos.view("Arc cosine of " + String(nuUserVal))
   else
      msgStop("You entered: " + String(nuUserVal), stPrompt)
   endIf
endMethod
```

asin method

Returns the 2-quadrant arc sine of a number.

Syntax

asin ( ) Number

Description

Given a number between -1 and 1, **asin** returns a numeric value between -pi/2 and pi/2, expressed in radians. **asin** is the inverse of **sin** if **asin**(*x*) = *y* and then **sin**(*y*) = *x*.

Example

{button ,AL(`OPAL_TYPE_NUMBER;OPAL_METH_NUACOS;OPAL_METH_NUATAN;OPAL_METH_NUCOS;OPAL_METH_NUSIN;',0,"Defaultoverview",)} Related Topics

asin example

In the following example, the **pushButton** method for the *findASin* button displays the arc sine of a number.

```
; findASin::pushButton
method pushButton(var eventInfo Event)
var
  x Number
endvar
x = .5
msgInfo("arc sine of .5", x.asin())   ; displays .52
endMethod
```

atan method

Returns the 2-quadrant arctangent of a number.

Syntax

atan ( ) Number

Description

Given a tangent in radians, **atan** returns the angle in radians. **atan** is called the 2-quadrant arctangent because it returns values within quadrants 1 and 4 (i.e., between -pi/2 and pi/2). **atan** is the inverse of **tan** if atan(*x*) = *y* and then tan(*y*) = *x*.

Example

{button ,AL(`OPAL_TYPE_NUMBER;OPAL_METH_NUCOS;OPAL_METH_NUSIN;OPAL_METH_NUTAN;OPAL_METH_NUTANH;OPAL_METH_NUATAN2;',0,"Defaultoverview",)} Related Topics

atan example

In the following example, the **pushButton** method for *getAtan* calculates the 2-quadrant arctangent of *x* and *y* :

```
; getAtan::pushButton
method pushButton(var eventInfo Event)
var
  x   Number
  checkPi, fortyFiveDegrees Number
endvar
x = 1
fortyFiveDegrees = x.atan()
msgInfo("45 degrees in radians: ", fortyFiveDegrees)  ; 0.79
checkPi = fortyFiveDegrees * 4       ; pi radians = 180 degrees
msgInfo("pi: ", format("w12.10", checkPi))
endMethod
```

atan2 method

Returns the 4-quadrant arctangent of a number.

Syntax

```
atan2 ( const x Number ) Number
```

Description

Given a sine in radians, **atan2** returns an angle in radians with cosine x. **atan2** is called the 4-quadrant arctangent because it returns values in all four quadrants.

Example

{button ,AL(`OPAL_TYPE_NUMBER;OPAL_METH_NUCOS;OPAL_METH_NUSIN;OPAL_METH_NUTAN;OPAL_METH_NUTANH;',0,"Defaultoverview",)} Related Topics

atan2 example

The following example assumes that a form contains a button named *getAtan2*. The **pushButton** method for *getAtan2* calculates the 4-quadrant arctangent of *x* and *y* and then displays the results:

```
; getAtan2::pushButton
method pushButton(var eventInfo Event)
var
  x,
  y,
  checkpi,
  fortyFiveDegrees Number
endvar
x = 1                          ; The angle whose tangent is 1 / 1
y = 1                          ; is a 45 degree angle
fortyFiveDegrees = x.atan2(y)
msgInfo("45 degrees in radians: ", fortyFiveDegrees)  ; 0.79
checkpi = fortyFiveDegrees * 4.0      ; pi radians = 180 degrees
msgInfo("pi: ", format("w12.10", checkpi))
endMethod
```

ceil method

Rounds a numeric expression up to the nearest whole number.

Syntax

ceil ( ) Number

Description

ceil rounds a numeric expression up (toward positive infinity) to the nearest whole number.

Example

{button ,AL(`OPAL_TYPE_NUMBER;OPAL_METH_NUFLOR;',0,"Defaultoverview",)} **Related Topics**

ceil example

In the following example, the **pushButton** method for a button named *ceilVsRound* calculates the ceiling value of a number and then displays the rounded value of that number:

```
; ceilVsRound::pushButton
method pushButton(var eventInfo Event)
var
  x  Number
endVar
x = 3.1
msgInfo("The ceil of " + String(x) + " is", ceil(x))     ; displays 4.0
msgInfo("The round of " + String(x) + " is", x.round(0)) ; displays 3
endMethod
```

cos method

Returns the cosine of an angle.

Syntax

cos ( ) Number

Description

cos returns a value between -1 and 1 representing the cosine of an angle in radians.

Example

{button ,AL(`OPAL_TYPE_NUMBER;OPAL_METH_NUACOS;OPAL_METH_NUCOSH;OPAL_METH_NUSIN;OPAL_METH_NUTAN;',0,"Defaultoverview",)} Related Topics

cos example

In the following example, the **pushButton** method for the *findCosine* button calculates and displays the cosine of a 60-degree angle:

```
; findCosine::pushButton
method pushButton(var eventInfo Event)
var
  sixtyDegrees Number
endVar
sixtyDegrees = PI / 3.0
msgInfo("The cosine of 60 degrees", sixtyDegrees.cos()) ; displays 0.50
endMethod
```

cosh method

Returns the hyperbolic cosine of an angle.

Syntax

cosh ( ) Number

Description

cosh returns the hyperbolic cosine of an angle in radians. **cosh** uses the following formula:

cosh(angle) = (exp(angle) + exp(-angle)) / 2

Example

{button ,AL(`OPAL_TYPE_NUMBER;OPAL_METH_NUCOS;OPAL_METH_NUSIN;OPAL_METH_NUTAN;',0,"Defaultoverview",)} Related Topics

cosh example

The following example uses the **pushButton** method for the *findCosineH* button to calculate and display the hyperbolic cosine of a 60 degree angle:

```
; findCosineH::pushButton
method pushButton(var eventInfo Event)
var
  sixtyDegrees Number
endVar
sixtyDegrees = PI / 3.0
msgInfo("The h cosine of " + format("W8.6", sixtyDegrees) + " radians",
         format("W14.12", sixtyDegrees.cosh()))
; displays 1.600286857702
endMethod
```

exp method

Returns the exponential (base *e*) of a number.

Syntax

exp ( ) Number

Description

exp computes *e* to the *x* power, where the constant *e* is 2.7182845905 (the so-called natural number), and the return value is the exponent x. The inverse method is the natural log, **ln**.

Example

{button ,AL(`OPAL_TYPE_NUMBER;OPAL_METH_NULN;OPAL_METH_NULOG;',0,"Defaultoverview",)}

Related Topics

exp example

In the following example, the **pushButton** method for a button named *getExponent* button calculates and displays the base *e* of 1:

```
; getExponent::pushButton
method pushButton(var eventInfo Event)
msgInfo("The exp of 1.0", format("W14.12", exp(1.0)))
; exp(1) formatted to display full precision
endMethod
```

floor method

Rounds a numeric expression down to the nearest whole number.

Syntax

floor ( ) Number

Description

floor rounds a numeric expression down (toward negative infinity) to the nearest whole number.

Example

{button ,AL(`OPAL_TYPE_NUMBER;OPAL_METH_NUCEIL;',0,"Defaultoverview",)} **Related Topics**

floor example

In the following example, the **pushButton** method for a button named *floorVsRound* uses **floor** to round *x* down to the nearest integer. By comparison, for the same number, **round** results in a higher number.

```
; floorVsRound::pushButton
method pushButton(var eventInfo Event)
var
  x  Number
endVar
x = 3.9
msgInfo("The floor of " + String(x) + " is", floor(x))   ; displays 3.0
msgInfo("The round of " + String(x) + " is", x.round(0)) ; displays 4.0
endMethod
```

fraction method

Returns the fractional portion of a number.

Syntax

fraction ( ) Number

Description

fraction returns the fractional portion of a number (i.e., the part to the right of the decimal).

Example

{button ,AL(`OPAL_TYPE_NUMBER;OPAL_METH_NUMOD;',0,"Defaultoverview",)} **Related Topics**

fraction example

In the following example, the **pushButton** method for *fractButton* displays the fraction portion of a numeric variable:

```
; fractButton::pushButton
method pushButton(var eventInfo Event)
var
  myNum Number
endVar
myNum = 12.23
msgInfo("Fractional part of " + String(myNum),
         myNum.fraction())                  ; displays .23
endMethod
```

fv method

Returns the future value of a series of equal payments.

Syntax

```
fv ( const interestRate Number, periods Number ) Number
```

Description

fv returns the future value of a series of equal payment periods, invested at an interest rate specified by *interestRate*. *interestRate* is expressed as a decimal number. Ensure that the rate period matches the deposit period (i.e., if the deposits are monthly, the interest rate is also monthly).

fv uses the following formula:

FV = payment(pow(1 + *rate*, *periods*) - 1) / *rate*

fv is also called the future or compound value of an annuity because it calculates the amount accumulated in an annuity fund when making regular, equal payments over time.

Example

{button ,AL(`OPAL_TYPE_NUMBER;OPAL_METH_NUPMT;OPAL_METH_NUPV;',0,"Defaultoverview",)}
Related Topics

fv example

The following example calculates how much a 14.5% Individual Retirement Account is worth if $166.67 is deposited each month for 30 years.

```
; findFutureVal::pushButton
method pushButton(var eventInfo Event)
var
  depositAmt,
  intRate,
  numPayments,
  iraValue       Number
endVar
intRate = .145 / 12      ; convert yearly interest to monthly interest
numPayments = 360        ; monthly payments for 30 years
depositAmt = 166.67      ; monthly deposit amount ($2000 a year)
iraValue = depositAmt.fv(intRate, numPayments)
msgInfo("IRA Value", "Depositing " + String(depositAmt) +
        " a month for " + String(numPayments/12) + " years at " +
        String(intRate * 12 * 100) + "% yields " + String(iraValue) +
        ". You'll be old but you'll be rich!")
; displays "Depositing 166.67 a month for 30 years
;           at 14.50% yields 1,027,394.23 ..."
endMethod
```

ln method

Returns the natural logarithm of a numeric expression.

Syntax

```
ln ( ) Number
```

Description

ln calculates the natural logarithm to the base e of a positive value. The constant *e* is the natural number, approximated by the value 2.7182845905. If the specified value is 0 or negative, **ln** fails.

The inverse method is **exp**. Use **log** to compute base 10 logarithms.

Example

{button ,AL(`OPAL_TYPE_NUMBER;OPAL_METH_NUEXP;OPAL_METH_NULOG;',0,"Defaultoverview",)}

Related Topics

In example

In the following example, the **pushButton** method for the *findNatLog* button calculates and displays the natural logarithm of several numbers:

```
; findNatLog::pushButton
method pushButton(var eventInfo Event)
var
  x  Number
endVar
x = 2.71828
msgInfo("Natural log of " + Format("W10.6", x), ln(x)) ; displays 1.00
x = 7.3891
msgInfo("Natural log of " + Format("W10.6", x), ln(x)) ; displays 2.00
x = 20.0855
msgInfo("Natural log of " + Format("W10.6", x), ln(x)) ; displays 3.00
endMethod
```

log method

Returns the base 10 logarithm of a numeric expression.

Syntax

log ( ) Number

Description

log returns the base 10 logarithm of a value or numeric expression. If the specified value is 0 or negative, **log** fails.

Use **ln** to compute natural logarithms.

Example

{button ,AL(`OPAL_TYPE_NUMBER;OPAL_METH_NUEXP;OPAL_METH_NULN;',0,"Defaultoverview",)}

Related Topics

log example

The following example uses the a button's **pushButton** method to calculate and display the base 10 logarithm of a value.

```
; findLog::pushButton
method pushButton(var eventInfo Event)
var
  x Number
endVar
x = 10
msgInfo("The logarithm of " + String(x), log(x)) ; displays 1.00
x = 100
msgInfo("The logarithm of " + String(x), log(x)) ; displays 2.00
x = 1000
msgInfo("The logarithm of " + String(x), log(x)) ; displays 3.00
endMethod
```

max procedure

Returns the larger of two numbers.

Syntax

```
max ( const x1 AnyType, const x2 AnyType ) AnyType
```

Description

max returns the larger of two values specified by *x1* and *x2*.

Example

{button ,AL(`OPAL_TYPE_NUMBER;OPAL_METH_NUMIN;',0,"Defaultoverview",)} Related Topics

max example

The following example, calculates a medical deduction for tax purposes. The **pushButton** method for *findMedDeduct* calculates the maximum of 7.5% of *AGI* or *medExpense* and then deducts 7.5% of *AGI* from the result. Finding the maximum number first ensures that the calculation returns a positive number.

```
; findMedDeduct
method pushButton(var eventInfo Event)
var
  medExpense,
  AGI        Number
endVar
AGI = 32000.45
medExpense = 4035.24
msgInfo("Allowed Medical Deduction",
        max(medExpense, AGI * .075) - (AGI * .075))  ; displays 1,635.21
; assumes that you can deduct only that part of your medical and dental
; expenses greater than 7.5% of Adjusted Gross Income
endMethod
```

min procedure

Returns the smaller of two numbers.

Syntax

```
min ( const x1 AnyType, const x2 AnyType ) AnyType
```

Description

min returns the smaller of two values specified by *x1* and *x2*.

Example

min example

The following example calculates the maximum amount of tax-deductible charitable contributions when no more than 30% of the adjusted gross income can be deducted. The **pushButton** method for the *findCharityDeduct* button calculates and displays the minimum of 30% of *AGI* and *charity*.

```
; findCharityDeduct::pushButton
method pushButton(var eventInfo Event)
var
  charity,
  AGI       Number
endVar
AGI = 32000.45     ; Adjusted Gross Income
charity = 12000    ; charitable contributions for the year
msgInfo("Allowed Charity Deduction", min(charity, AGI * .30))
; displays 9,600.13
; assumes charitable contributions up to 30% of AGI
; are allowed as deductions
endMethod
```

mod method

Returns the remainder when one number is divided by another.

Syntax

```
mod ( const modulo Number ) Number
```

Description

mod returns the remainder (or modulus) when one number is divided by the value of *modulo*. If the number is greater than the value of *modulo*, **mod** returns the remainder. If the number is less than *modulo*, **mod** returns the number. If the number equals *modulo*, **mod** returns 0. The following table illustrates each scenario:

Fraction	ObjectPAL code	Return value
5/2	num = 5	um.mod(2) 1
2/5	num = 2	um.mod(5) 2
2/2	num = 2	um.mod(2) 0

Example

{button ,AL(`OPAL_TYPE_NUMBER;OPAL_METH_NUFRACTION;',0,"Defaultoverview",)} Related Topics

mod example

In the following example, the **pushButton** method for the *showRemainder* button calculates and displays the modulus for a series of division operations:

```
; showRemainder::pushButton
method pushButton(var eventInfo Event)
var
  x Number
endVar
x = 8
msgInfo("The remainder of " + String(x) + "/" + "3",
          x.mod(3))                    ; displays 2
msgInfo("The remainder of " + String(x) + "/" + "12",
         x.mod(12))                    ; displays 8
x = -2
msgInfo("The remainder of " + String(x) + "/" + "10",
          x.mod(10))                   ; displays -2
x = -10
msgInfo("The remainder of " + String(x) + "/" + "-100",
        x.mod(-100))                   ; displays -10
endMethod
```

number procedure

Casts a value as a Number.

Syntax

```
number ( const value AnyType ) Number
```

Description

number casts *value* to a Number. *value* must be in the form of a valid number that can be entered in a field. When a numeric operand is required in an expression, or when a numeric argument is required in a procedure or method, **number** is used to cast a non-numeric type to a Number. **number** behaves the same as **numVal**.

Example

{button ,AL(`OPAL_TYPE_NUMBER;OPAL_METH_NUNVAL;',0,"Defaultoverview",)} Related Topics

number example

In the following example, a variable *x* is declared as a String and then assigned a string of numbers. The **pushButton** method for the *showDouble* button casts *x* to a Number before doubling it:

```
; showDouble::pushButton
method pushButton(var eventInfo Event)
var
  x  String
endVar
x = "1123.54"
; cast x to a Number before multiplying by 2
msgInfo("Double " + x + " is", Number(x) * 2) ; displays 2,247.08
endMethod
```

numVal procedure

Casts a value as a Number.

Syntax

```
numVal ( const value AnyType ) Number
```

Description

numVal casts *value* to a Number. *value* must be in the form of a valid number that can be entered in a field. **numVal** is most often used to cast a non-numeric type to a Number when a numeric operand is required in an expression, or a numeric argument is required in a procedure or method. **numVal** behaves the same as **number**.

Example

{button ,AL(`OPAL_TYPE_NUMBER;OPAL_METH_NUNUMB;',0,"Defaultoverview",)} Related Topics

numVal example

In the following example, a variable *x* is declared as a String and then assigned a string of numbers. The **pushButton** method for the *showDouble* button casts *x* to a Number before doubling it:

```
; showDouble::pushButton
method pushButton(var eventInfo Event)
var
  x  String
endVar
x = "1123.54"
; cast x to a Number before multiplying by 2
msgInfo("Double " + x + " is", numVal(x) * 2) ; displays 2,247.08
endMethod
```

pmt method

Returns the periodic payment required to pay off a loan.

Syntax

```
pmt ( const interestRate Number, const periods Number ) Number
```

Description

pmt returns the constant, regular payment required to pay off a loan. **pmt** uses the following formula:

PMT = *p* * *i* / (1 - (1 + *i*) ^-t)

(where *p* = principal amount, *i* = effective interest rate per period, and *t* = term of the loan or number of payment periods).

Payments are due at the end of each period.

pmt works for amortization-type loans (e.g., conventional home mortgages), in which part of the payment consists of interest on the remaining principal, and the remainder pays off part of the principal of the loan. **pmt** does not work for consumer-type loans (e.g., repayments of credit accounts or automobile loans).

The interest rate used in **pmt** is a decimal number. Ensure that the rate period matches the payment periods (i.e., if the payments are monthly, the interest rate should also be monthly). Because the interest rate for amortization loans (mortgages) is usually annual, you can divide it by 12 for monthly payments or by 4 for quarterly payments.

Use the nominal annual interest rate quoted instead of the accompanying annual percentage rate (APR).

Example

{button ,AL(`OPAL_TYPE_NUMBER;OPAL_METH_NUFV;OPAL_METH_NUPV;',0,"Defaultoverview",)}

Related Topics

pmt example

In the following example, the **pushButton** method for the *findPayment* button calculates the monthly payment for a 24-month loan of $1,000 at a 12% interest rate:

```
; findPayment::pushButton
method pushButton(var eventInfo Event)
var
  monthlyPayment,
  loanAmt,
  intRate,
  numPayments Number
endVar
loanAmt = 1000        ; borrow $1000
intRate = .12 / 12    ; 12 percent annual interest
numPayments = 24      ; 1 payment per month for 2 years
monthlyPayment = loanAmt.pmt(intRate, numPayments)
msgInfo("Monthly payment", "The monthly payment for a loan of " +
        String(loanAmt) + " at " + String(intRate * 12 * 100) +
        "% interest for " + String(SmallInt(numPayments)) +
        " months is " + String(monthlyPayment))    ; payment is $47.07
endMethod
```

pow method

Raises a number to a specified power.

Syntax

```
pow ( const exponent Number ) Number
```

Description

pow returns the value of a number raised to the power specified in *exponent*. If the return value is larger than 1E308 or smaller than 1E-308, **pow** returns an error.

Example

{button ,AL(`OPAL_TYPE_NUMBER;OPAL_METH_NUPOW10;OPAL_METH_NULN;OPAL_METH_NULOG;',0,"Defaultoverview",)} Related Topics

pow example

In the following example, the **pushButton** method for the *raiseTwo* button calculates $root^{expn}$ and displays the result:

```
; raiseTwo::pushButton
method pushButton(var eventInfo Event)
var
  root,
  expn    Number
endVar
root = 2
expn = 8
msgInfo(String(root) + " raised to the power of " +
        String(expn), root.pow(expn)) ; displays 256
endMethod
```

pow10 method

Calculates 10 to a specified power.

Syntax

pow10 ( ) Number

Description

pow10 returns the value of 10 raised to a specified power.

Example

{button ,AL(`OPAL_TYPE_NUMBER;OPAL_METH_NUPOW;OPAL_METH_NULN;OPAL_METH_NULOG;',0,"Defaultoverview",)} Related Topics

pow10 example

In the following example, the **pushButton** method for the *raiseTen* button calculates 10^{expn} and displays the result:

```
; raiseTen::pushButton
method pushButton(var eventInfo Event)
var
  expn,
  result Number
endVar
expn = 9
result = expn.pow10()
msgInfo("Ten raised by a power of " + String(expn),
        format("EC", result))                  ; displays 1,000,000,000
endMethod
```

pv method

Returns the current value of a series of equal payments.

Syntax

```
pv ( const interestRate Number, const periods Number ) Number
```

Description

pv calculates the current value of equal, regular payments on a loan (or withdrawals from an investment) at a rate specified in *interestRate* for a term specified in *periods*. The payments reduce the principal, but the remaining balance continues to generate and compound interest.

pv uses the following formula:

```
PV = payment * (1- (1 +rate)^-n / rate)
```

(where *n* is the number of periods)

The interest rate used in **pv** is expressed as a decimal number. Ensure that the rate period matches the payment period (i.e., if the payments are monthly, the interest rate should also be monthly). Use **pv** to calculate the size of the mortgage you can afford. (Use **pmt** to work in reverse and find the monthly payment needed to amortize a given amount.) You can also use **pv** to calculate the amount you'll need to purchase an annuity that makes regular, equal payments to you over time. For this reason, **pv** is also called the present value of an annuity.

Example

{button ,AL(`OPAL_TYPE_NUMBER;OPAL_METH_NUFV;OPAL_METH_NUPMT;',0,"Defaultoverview",)}

Related Topics

pv example

The following example assumes that you can afford to pay $1,200 per month and can get a 30-year mortgage at a fixed annual rate of 9% (0.75% monthly). The **pushButton** method for *findPV* calculates and displays the loan amount for which you qualify:

```
; findPV::pushButton
method pushButton(var eventInfo Event)
var
  payAmt,
  intRate,
  term,
  mortgage    Number
endVar
payAmt   = 1200
intRate  = .09 / 12             ; monthly interest for 9% a year
term     = 360                  ; 30 years (expressed in months)
mortgage = payAmt.pv(intRate, term)
msgInfo("Maximum Mortgage", "If you can pay " + String(payAmt) +
        " a month for " + String(term /12) + " years at " +
        String(intRate * 12 * 100) + "% you can qualify for " +
        format("E$C", mortgage))     ; displays $149,138
endMethod
```

Imagine when you retire you would like to withdraw $2,500 each month for 30 years from an annuity account that accumulates 7.5% annual interest. This code uses the **pushButton** method for the *findAnnuity* button to calculate how much you'll need in the account:

```
; findAnnuity::pushButton
method pushButton(var eventInfo Event)
var
  monthlyAmt,
  term,
  intRate,
  investment  Number
endVar


monthlyAmt = 2500.00  ; monthly amount you want annuity to pay
term = 360            ; 30 years, converted to 360 months
intRate = .075/12     ; 7.5% a year, converted to monthly rate
investment = monthlyAmt.pv(intRate, term)  ; what you need to start with
msgInfo("Annuity Required", "For an annuity to return $" +
        String(monthlyAmt) + " a month at " +
        format("W4.2", intRate * 12 * 100) + "% for " +
        String(SmallInt(term / 12)) +
        " years, the original amount must be " +
        String(investment))                ; displays 357,544.07
endMethod
```

rand procedure

Generates a random value ranging from 0 to 1.

Syntax

rand ( ) Number

Description

rand generates a random value ranging from 0 to 1.

Example

{button ,AL(`OPAL_TYPE_NUMBER;OPAL_METH_NUTRUNCATE;',0,"Defaultoverview",)} **Related Topics**

rand example

In the following example, the **pushButton** method for the *getRand* button calculates and displays a random number *x* between 1 (*minNum*) and 10 (*maxNum*).

```
; getRand::pushButton
method pushButton(var eventInfo Event)
var
  x,
  minNum,
  maxNum  SmallInt
endVar
minNum = 1
maxNum = 10
; get a random integer between minNum and maxNum
x = SmallInt(rand() * (maxNum - minNum + 1) + minNum)
msgInfo("A number between " + String(minNum) + " and " +
        String(maxNum),  x)
endMethod
```

round method

Rounds a number to a specified number of decimal places.

Syntax

```
round ( const places SmallInt ) Number
```

Description

round returns a number rounded to the number of decimal places specified in *places*.

Example

{button ,AL(`OPAL_TYPE_NUMBER;OPAL_METH_NUTRUNCATE;OPAL_METH_NUCEIL;OPAL_METH_NUFLOR;',0,"Defaultoverview",)} Related Topics

round example

In the following example, the **pushButton** method for the *showRound* button rounds a number to 4 decimal places and displays the result. This code then rounds and displays a number to the nearest 1000.

```
; showRound::pushButton
method pushButton(var eventInfo Event)
var
  roundMe Number
endVar
roundMe = 1.2356838
msgInfo(format("W9.7",roundMe) + " rounded to 4 decimal places",
          format("W6.4", roundMe.round(4))) ; displays 1.2357
roundMe = 678394
msgInfo(String(roundMe) + " rounded to -3 decimal places",
          roundMe.round(-3))               ; displays 678,000
endMethod
```

sin method

Returns the sine of an angle.

Syntax

```
sin ( ) Number
```

Description

sin returns a number between -1 and 1 representing the sine of an angle in radians.

Example

{button ,AL(`OPAL_TYPE_NUMBER;OPAL_METH_NUASIN;OPAL_METH_NUCOS;OPAL_METH_NUTAN;',0,"Defaultoverview",)} **Related Topics**

sin example

The following example uses the **pushButton** method for the *findSin* button to calculate the sine of a 45-degree angle:

```
; findSin::pushButton
method pushButton(var eventInfo Event)
var
  fortyFiveDegrees Number
endVar
fortyFiveDegrees  = PI / 4.0
msgInfo("The sine of 45 degrees",
         format("W14.12", fortyFiveDegrees.sin()))
; displays 0.707106781187
endMethod
```

sinh method

Returns the hyperbolic sine of an angle.

Syntax

sinh ( ) Number

Description

sinh returns the hyperbolic sine of an angle in radians. **sinh** uses the following formula:

sinh (*angle*) = (*exp* (*angle*) - *exp* (*-angle*)) / *2*

Example

{button ,AL(`OPAL_TYPE_NUMBER;OPAL_METH_NUSIN;OPAL_METH_NUCOS;OPAL_METH_NUTAN;',0,"Defaultoverview",)} Related Topics

sinh example

In the following example, the **pushButton** method for the *getHSine* button calculates the hyperbolic sine of a 45-degree angle:

```
; getHSine
method pushButton(var eventInfo Event)
var
  fortyFiveDegrees Number
endVar
fortyFiveDegrees = PI / 4.0
msgInfo("The hyperbolic sine of 45 degrees",
        format("w14.12", fortyFiveDegrees.sinh()))
; displays 0.868670961486
endMethod
```

sqrt method

Returns the square root of a number.

Syntax

```
sqrt ( ) Number
```

Description

sqrt returns the square root of a positive value or numeric expression.

Example

{button ,AL(`OPAL_TYPE_NUMBER;OPAL_METH_NUEXP;',0,"Defaultoverview",)} Related Topics

sqrt example

In the following example, the **pushButton** method for the *getSqrt* button assigns the value from *fieldOne* (an unbound field object) to *x*. If *x* is positive, the code then calculates and displays the square root of *x*:

```
; getSqrt::pushButton
method pushButton(var eventInfo Event)
var
  x Number
endVar
x = fieldOne
if x < 0 then
  msgStop("Sorry",
          "Can't take the square root of a negative number.")
else
  msgInfo("The square root of " + String(x),
           format("w14.6", sqrt(x))) ; displays result
endIf
endMethod
```

tan method

Returns the tangent of an angle.

Syntax

tan ( ) Number

Description

tan returns the tangent of an angle in radians. **tan** diverges at -pi/2, pi/2, and every ± pi radians from those values.

Example

{button ,AL(`OPAL_TYPE_NUMBER;OPAL_METH_NUATAN;OPAL_METH_NUATAN2;OPAL_METH_NUCOS;OPAL_METH_NUSIN;',0,"Defaultoverview",)} Related Topics

tan example

In the following example, the **pushButton** method for the *getTan* button calculates the tangent of a 45-degree angle and displays the result:

```
; getTan::pushButton
method pushButton(var eventInfo Event)
var
  fortyFiveDegrees Number
endVar
fortyFiveDegrees = PI / 4.0
msgInfo("Tangent of 45 degrees", fortyFiveDegrees.tan())   ; displays 1.00
endMethod
```

tanh method

Returns the hyperbolic tangent of an angle.

Syntax

tanh ( ) Number

Description

tanh returns the hyperbolic tangent of an angle in radians. **tanh** uses the following formula:

tanh (angle) = sinh (angle) / cosh (angle)

Example

{button ,AL(`OPAL_TYPE_NUMBER;OPAL_METH_NUATAN;OPAL_METH_NUCOS;OPAL_METH_NUSIN;',0,"Defaultoverview",)} Related Topics

tanh example

In the following example, the **pushButton** method for a button named *getHTan* calculates the hyperbolic tangent of a 60-degree angle and displays the result:

```
; getHTan::pushButton
method pushButton(var eventInfo Event)
var
  sixtyDegrees Number
endVar
sixtyDegrees = PI / 3.0
msgInfo("The hyperbolic tangent of 60 degrees",
        format("W14.12", sixtyDegrees.tanh()))
; displays .780714435359
endMethod
```

truncate method

Shortens a number to a specified number of decimal places.

Syntax

```
truncate ( const places SmallInt ) Number
```

Description

truncate returns a number truncated toward 0 to the number of decimal places specified in *places*. **truncate** does not round the value.

Example

{button ,AL(`OPAL_TYPE_NUMBER;OPAL_METH_NUCEIL;OPAL_METH_NUFLOR;OPAL_METH_NUROUND;',0, "Defaultoverview",)} **Related Topics**

truncate example

In the following example, the **pushButton** method for the *chopAValue* button assigns the value from *fieldOne* (an unbound field object) to *x*. the code then truncates *x* to 3 decimal places, and displays the truncated result:

```
; chopAValue::pushButton
method pushButton(var eventInfo Event)
var
  x Number
endVar
x = fieldOne
msgInfo("x truncated to 3 places",
        format("W14.6", x.truncate(3))) ; displays truncated version of x
endMethod
```

OLE type

Object Linking and Embedding (OLE) is a protocol that allows you to access another application without leaving Corel Paradox.

For example, suppose you have tables that contain bitmap graphics, and you want to create a Corel Paradox application that enables users to edit those graphics. One approach is to create the graphics using a paint program that is an OLE server (defined below). Then, use ObjectPAL OLE type methods to make the functionality of the paint program available to your users (assuming, of course, that your users have the paint program installed on their systems).

ObjectPAL and Corel Paradox also support Dynamic Data Exchange (DDE) another protocol that allows you to share data among applications.

The following terms are used when discussing OLE operations:

OLE *server*	An application that uses the OLE mechanism to provide access to its documents. Corel Paradox is an OLE server.
OLE *container*	An application that uses the OLE mechanism to access documents created by an OLE server. Corel Paradox is an OLE container.
OLE *object*	A document created using an OLE server. A document that contains the data you want to use in your Corel Paradox application.
OLE *variable*	An ObjectPAL variable declared as an OLE type. An OLE variable provides a handle for manipulating an OLE object. You can use OLE variables in ObjectPAL code to manipulate OLE objects.
Asynchronous	Code in each application executes independently (i.e., one application does not wait for the other). When you use a method that launches an OLE server for user input, declare the OLE variable in a Var window or in a method window above the **method** keyword. This ensures that the OLE variable is and in scope, even if the method finishes before the server application is closed.

The following table lists the methods for the OLE type, including several derived methods from the AnyType type.

Methods for the OLE type

AnyType	**OLE**
blank	**canLinkFromClipboard**
dataType	**canReadFromClipboard**
isAssigned	**edit**
isBlank	**enumServerClassNames**
isFixedType	**enumVerbs**
unAssign	**getServerName**
	insertObject
	isLinked
	linkFromClipboard
	readFromClipboard
	updateLinkNow
	writeToClipboard

Print related ObjectPAL methods and examples

canLinkFromClipboard method

Reports whether an OLE object can be linked from the Clipboard to an OLE variable.

Syntax

canLinkFromClipboard ( ) Logical

Description

canLinkFromClipboard returns True if an OLE object can be linked from the Clipboard to an OLE variable; otherwise, it returns False. After an OLE object is linked from the Clipboard, changes made to the OLE object, while in Corel Paradox, affect the underlying file.

canLinkFromClipboard is useful in a routine that determines whether a **linkFromClipboard** operation is possible. A menu item is dimmed and inactive when **canLinkFromClipboard** returns False.

Example

{button ,AL(`OPAL_TYPE_OLE;OPAL_METH_OLLINKFROMCLIPBOARD;OPAL_METH_OLCANREADFROM;OPAL_METH_OLREADFROM;',0,"Defaultoverview",)} Related Topics

canLinkFromClipboard example

The following example attempts to link an OLE object from the Clipboard to a field in a specified record in a table. If the OLE object can't be linked, this code prompts the user to embed or read the OLE object instead.

```
; btnLinkOrRead::pushButton
method mouseClick(var eventInfo MouseEvent)
   var
      stReadOLE      String
      oleObj         OLE
      tcEmployee      TCursor
   endVar


   ; Move to specified record in table.
   tcEmployee.open("employee")
   tcEmployee.locate("EmpName", "Frank Corel")


   ; Link if you can, otherwise read (embed).
   switch
      case oleObj.canLinkFromClipboard() :
         oleObj.linkFromClipboard()


      case oleObj.canReadFromClipboard() :
         stReadOLE = msgQuestion("Can't link OLE object.",
                                 "Do you want to embed it instead?")
         if stReadOLE = "Yes" then
            oleObj.readFromClipboard()
         else
            message("No update.")
            return
         endIf

      otherwise :
         msgInfo("Can't link or embed the OLE object.",
              "The Clipboard may be empty.")
         return
   endSwitch


   ; Update the table.
   tcEmployee.edit()
   tcEmployee.VoiceSample = oleObj
   tcEmployee.endEdit()
   message("Update complete")
endMethod
```

canReadFromClipboard method

Reports whether an OLE object can be embedded from the Clipboard to an OLE variable.

Syntax

canReadFromClipboard ( ) Logical

Description

canReadFromClipboard returns True if an OLE object can be embedded or read from the Clipboard into an OLE variable; otherwise, it returns False. After an OLE object is read from the Clipboard, changes made to the OLE object while in Corel Paradox, do not affect the underlying file.

canReadFromClipboard is useful in a routine that determines whether a **readFromClipboard** operation is possible. A menu item is dimmed and inactive when **canReadFromClipboard** returns False.

Example

{button ,AL(`OPAL_TYPE_OLE;OPAL_METH_OLREADFROM;OPAL_METH_OLLINKFROMCLIPBOARD;OPAL_METH_OLCANLINKFROMCLIPBOARD;',0,"Defaultoverview",)} Related Topics

canReadFromClipboard example

See the **canLinkFromClipboard** example.

edit method

Launches the OLE server and allows the user to edit the object or perform another action.

Syntax

```
edit ( const oleText String, const verb SmallInt ) Logical
```

Description

edit launches the OLE server application and gives control to the user. The argument *oleText* is a string that Corel Paradox passes to the server application. Many server applications displays *oleText* in the Title Bar. **edit** passes *verb* to the application server to specify an operation.

verb is an integer that corresponds to one of the OLE server's action constants. The meaning of *verb* varies from application to application a *verb* that is appropriate for one application may not be appropriate for another. Use **enumVerbs** to determine which verbs the server supports and then select a verb for the call to **edit**.

If you want to launch an OLE server without using **enumVerbs**, use 0 for *verb* this value represents the primary verb, and should be supported by all OLE servers.

Example

{button ,AL(`OPAL_TYPE_OLE;OPAL_METH_OLENUMVERBS;',0,"Defaultoverview",)} **Related Topics**

edit example

The following example assumes that the *Pics* table stores Paintbrush graphics in an OLE field. The table has two fields: PicName (A8) and PicData (O). When you click *editButton*, this code locates a record in the table and uses **edit** to invoke Paintbrush (enabling the user to edit the graphic in the OLE field). When you click *updateButton*, the code updates the *Pics* table.

Code is attached to the page's Var window, to the *editButton*'s **pushButton** method, and to the *updateButton*'s **pushButton** method. Variables are declared in the page's Var window for two reasons: to make them available to both buttons; it ensures the OLE variable is available, even if **edit** finishes executing before Paintbrush is closed.

The page's Var window contains the following code:

```
 var
   olePic  OLE
   picTC   TCursor
endVar
```

The *editButton*'*s* **pushButton** method contains the following code:

```
 method pushButton(var eventInfo Event)
   if picTC.open ("pics.db") then
      if picTC.locate("PicName", "blueLine") then


                  ; The PicData field stores OLE objects
                  ; created using Paintbrush.
         olePic = picTC.PicData


                  ; Launch Paintbrush so user can edit the bitmap.
         olePic.edit("PDOXWIN", 0)
      else
         msgStop("Stop", "Couldn't find blueLine.")
      endIf
   else
      msgStop("Stop", "Couldn't open table.")
   endIf
endMethod
```

The *updateButton*'*s* **pushButton** method contains the following code:

```
 method pushButton(var eventInfo Event)
   picTC.edit()
   picTC.PicData = olePic
   picTC.endEdit()
   picTC.close()
endMethod
```

enumServerClassNames method

Lists the registered OLE servers.

Syntax

```
enumServerClassNames ( var serverClasses DynArray[ ] String ) Logical
```

Description

enumServerClassNames lists the OLE servers registered on the user's system. The information is assigned to *serverClasses*, a dynamic array (DynArray) that you must declare and pass as an argument. This method returns True if it succeeds; otherwise, it returns False.

The DynArray's indexes are the end-user server names (e.g., Corel Paradox Table), and the corresponding items are the internal OLE names.

Use **enumServerClassNames** to pass a server name to **insertObject**.

Example

{button ,AL(`OPAL_TYPE_OLE;OPAL_METH_OLENUMVERBS;OPAL_METH_OLINSERTOBJECT;',0,"Defaultoverview",)} Related Topics

enumServerClassNames example

See the **insertObject** example.

enumVerbs method

Lists the actions supported by an OLE server.

Syntax

```
enumVerbs ( var verbs DynArray[ ] SmallInt ) Logical
```

Description

enumVerbs creates a dynamic array (DynArray) listing the action commands or *verbs* supported by the OLE server associated with an OLE variable.

When you associate an OLE variable with an OLE object, Corel Paradox recognizes the server application which generated the object. OLE methods like **enumVerbs** and **getServerName** allow you to ask questions.

enumVerbs requests the server for a list of supported verbs and then loads them into a DynArray. Each DynArray index corresponds to the name of a specific action (i.e., DynArray items correspond to the action constant used by the server). Because each verb's meaning varies from application to application, you must know which verb to pass to the server to instruct it to do what you want.

Windows Paintbrush is an OLE server that has only one action command (Edit, with a value of 0). The following code a Paintbrush graphic from the Clipboard and generates a dynamic array using **enumVerbs.** This code then displays the DynArray's contents in a dialog box.

```
var
  oleVar OLE
  dy DynArray[] SmallInt
endVar


oleVar.readFromClipboard() ; read from the Clipboard into oleVar
oleVar.enumverbs(dy)       ; generate a DynArray of verbs
dy.view()                  ; display DynArray contents in a dialog
```

This code assumes the Clipboard contains an OLE object (a graphic image) that was generated in Paintbrush. The dynamic array contains one element whose index is Edit and whose value is 0. Some OLE servers use more than one verb, and would therefore generate a larger list. Other OLE servers use Edit but preface the name with an ampersand (&Edit). The ampersand prefix is especially useful when you want to display action names in a menu. Corel Paradox recognizes the ampersand as a special character and displays &Edit as Edit. E is designated as an accelerator key.

For more information, see Menu methods.

Example

{button ,AL(`OPAL_TYPE_OLE;OPAL_METH_OLENUMSERVERCLASSNAMES;OPAL_METH_OLGETSERVERNAME;OPAL_METH_OLREADFROM;',0,"Defaultoverview",)} Related Topics

enumVerbs example

The following example assumes the *Sounds* table contains an alpha field named SoundName and an OLE field named SoundData. Data displayed in the OLE field is copied from the Windows Sound Recorder to the Clipboard. The following code uses **enumVerbs** to create a pop-up menu that lists the verbs (actions) for Sound Recorder when you click a button named *btnEditSounds*. Because Sound Recorder supports two actions (Edit and Play), this example allows the user to edit or play the sound contained in the OLE field.

The following code is attached to the button's Var window and declares the OLE variable. Declaring the OLE variable in the Var window, ensures that the variable is available, even if the method finishes before the server application is closed.

```
; btnEditSounds::Var
Var
   oleVar OLE
endVar
```

The following code is attached to the button's built-in **pushButton** method. It builds and displays a pop-up menu and launches the server application.

```
; btnEditSounds::pushButton
method pushButton(var eventInfo Event)
var
  oleVar  OLE
  p       PopUpMenu
  verbs   DynArray[] SmallInt
  tc      TCursor
  mChoice, tagName String
endvar
soundName = "tada.wav"
tblName = "Sounds.db"


if tc.open(tblName) then
  if tc.locate(1, soundName) then ; Search in first field for tada.wav
    oleVar = tc.SoundData    ; Assign field value to OLE var
    oleVar.enumVerbs(verbs)  ; Get list of Sound Recorder actions.
    forEach tagName in verbs ; Create a pop-up menu of verbs.
      p.addText(tagName) ; Sound Recorder's verbs are &Edit and &Play
    endForEach
    mChoice = p.show()  ; display "Edit" and "Play" in the pop-up menu


    ; If the user selects from the menu,
    ; pass the selected "verb" to the
    ; edit method. verbs[mChoice] evaluates to 0 or 1.
    ; "PdoxWin" appears in Sound Recorder's Title Bar
    ; when Edit is selected
    if not mChoice.isBlank() then
      oleVar.edit("PdoxWin", verbs[mChoice])
    endIf


  else
    errorShow("Can't find " + soundName + ".")
  endIf
else
  errorShow("Can't open " + tblName + ".")
endIf


endMethod
```

getServerName method

Reports the name of the OLE server for an OLE object.

Syntax

```
getServerName ( ) String
```

Description

getServerName reports the name of the OLE server for an OLE object. **getServerName** is especially useful when you want to provide the user with the OLE server name.

Example

{button ,AL(`OPAL_TYPE_OLE;OPAL_METH_OLEDIT;OPAL_METH_OLENUMSERVERCLASSNAMES;OPAL_METH_OLENUMVERBS;',0,"Defaultoverview",)} Related Topics

getServerName example

The following example assumes that the *Media* table has an alpha field named MediaName, an alpha field named ServerName, and an OLE field named MediaData. This code scans through *Media*'s records placing the name of the OLE server that generated data in the MediaData field.

```
; getServerName::pushButton
method pushButton(var eventInfo Event)
var
  oleVar  OLE
  tc      TCursor
endvar


if tc.open("Media") then
  tc.edit()
  scan tc for not isBlank(tc.SoundData) :
    oleVar = tc.SoundData
    tc.ServerName = oleVar.getServerName()
  endScan
  tc.close()
else
  msgStop("Error", "Can't open Media table.")
endIf


endMethod
```

insertObject method

Inserts a linked or embedded OLE object into an OLE variable.

Syntax

```
1. insertObject ( ) Logical
2. insertObject ( const fileName String , const link Logical ) Logical
3. insertObject ( const className String ) Logical
```

Description

insertObject assigns a linked or embedded OLE object to an OLE variable. This method returns True if it succeeds; otherwise, it returns False.

Syntax 1 invokes the Insert Object dialog box. The user must supply any necessary information and close the dialog box. For example, the user can choose Create New to insert a new OLE object or Create From File to insert an existing OLE object from a file.

Syntax 2 inserts an object from the file specified in *fileName* without launching the server application for user input. The argument *link* specifies whether to link to the file. If *link* is True, changes made to the object in Corel Paradox are reflected in the underlying file. If *link* is False, changes made in Corel Paradox do not affect the file.

Syntax 3 launches the server application for user input and inserts an object from the class specified in *className*. *className* is the name of a registered OLE server class. Use **enumServerClassNames** to view a list of OLE server class names.

Note

- When creating a new file, the server application may prompt the user for file creation information.

Examples

{button ,AL(`OPAL_TYPE_OLE;OPAL_METH_OLCANREADFROM;OPAL_METH_OLLINKFROMCLIPBOARD;OPAL_METH_OLREADFROM;',0,"Defaultoverview",)} Related Topics

insertObject method examples

Example1 Using variables declared in the Var window

Example2 Using **insertObject** without launching a server application

Example3 Using **insertObject** to launch a specified application

insertObject example 1

In the following example, a form contains buttons named *btnInsertOLE* and *btnEditOLE* and a field object named *mugShot*. *mugShot* is bound to an OLE field named MugShot in a table in the form's data model. The variables *oleVar* and *loInserted* are declared in the page's Var window to make them available to both buttons, and to ensure that the OLE variable is available if a method finishes before the server application is closed.

The following code is attached to the page's Var window. It declares the OLE variable named *oleVar* and a Logical flag variable named *loInserted* that tracks whether an OLE object was inserted into the OLE variable.

```
; thePage::Var
Var
   oleVar         OLE
   loInserted   Logical
endVar
```

The following code is attached to the **pushButton** method of *btnInsertOLE*. It displays the Insert Object dialog box, allowing the user to insert an OLE object:

```
; btnInsertOLE :: pushButton
method pushButton(var eventInfo Event)


   if not oleVar.insertObject() then ; Invoke Insert Object dialog box.
      errorShow()
      loInserted = No
      return
   else
      loInserted = Yes
   endIf


endMethod
```

The following code is attached to the **pushButton** method of *btnEditOLE*:

```
; btnEditOLE :: pushButton
method pushButton(var eventInfo Event)


   if not loInserted.isAssigned() then
      loInserted = No
   endIf


   if loInserted = Yes then
      edit()
      mugShot.Value = oleVar
      loInserted = No ; Reset the flag.
      endEdit()
   else
      msgInfo("No OLE object to insert.",
           "Click the Insert button.")
   endIf


endMethod
```

insertObject example 2

In the following example, a form contains a button named *btnInsertOLE* and a field object named *fldOLE*. *fldOLE* is bound to an OLE field in a table in the form's data model. The **pushButton** method uses an OLE variable *oleVar* and **insertObject** to read a wave file into the OLE variable named *oleVar*. The code then assigns the file to the field *fldOLE*. This example does not launch the server application for user input.

```
;btnInsertOLEFile :: pushButton
const
   ; Changes made in Corel Paradox will not
   ; affect the underlying file.
   kNoLink = False
endConst


var
   oleVar OLE
endVar


method pushButton(var eventInfo Event)
   var
      stFileName,
      stPrompt   String
   endVar


   stPrompt = "Type the filename here."
   stFileName = stPrompt
   stFileName.view("Enter a filename.")
   if stFileName = stPrompt then
      return ; User didn't type a filename and click OK.
   endIf

   if oleVar.insertObject(stFileName, kNoLink) then
      edit()
      fldOLE.Value = oleVar
      endEdit()
   else
      errorShow("Could not insert OLE object: " + stFileName)
   endIf
endMethod
```

insertObject example 3

Imagine that you are using Corel Paradox to maintain and publish a database for a school and each record represents a course syllabus. Since different instructors prefer different word processors, you can store syllabus data in an OLE field and let the instructors edit it any application that is an OLE server.

The following example assumes a form contains a table frame bound to the *Courses* table and that each record in the table frame contains a field object named *Syllabus*. The following code is attached to a button named *btnAddSyllabus* that allows the user to add a new syllabus to the table. This code displays a list of the OLE server applications installed in the user's system a in a pop-up menu. When the user chooses an application name from the pop-up menu, the call to **insertObject** inserts an object of the specified type.

```
; btnAddSyllabus :: pushButton
var
   oleVar   OLE
endVar


method pushButton(var eventInfo Event)
   var
      puServers      PopUpMenu
      stOLEServer,
      stUserServer      String
      dyOLEServers      DynArray[] String
   endVar


   ; Specify a title for the pop-up menu.
   puServers.addStaticText("Choose one:")
   puServers.addSeparator()


   ; enumServerClassNames returns a DynArray where the keys are
   ; the external names and the corresponding items are the
   ; names used internally by OLE.


   oleVar.enumServerClassNames(dyOLEServers)


   forEach stOLEServer in dyOLEServers
      puServers.addText(stOLEServer)
   endForEach


   stUserServer = puServers.show()
   if stUserServer <> "" then


     ; insertObject uses the internal name to specify an OLE server.
      if oleVar.insertObject(dyOLEServers[stUserServer]) then
         action(DataBeginEdit)
         Courses.Syllabus.Value = oleVar
         action(DataEndEdit)
      else
         errorShow("Could not insert " + stOLEServer)
      endIf
   else
      return ; User didn't choose a server.
   endIf
endMethod
```

isLinked method

Reports whether an OLE object is a linked object.

Syntax

isLinked ( ) Logical

Description

isLinked returns True if an OLE object is a linked object and False if it is an embedded object. When used with **updateLinkNow**, you can use this method to update the linked OLE fields in a table.

Example

{button ,AL(`OPAL_TYPE_OLE;OPAL_METH_OLCANREADFROM;OPAL_METH_OLREADFROM;OPAL_METH_OLUPDATELINKNOW;',0,"Defaultoverview",)} Related Topics

isLinked example

See the **updateLinkNow** example.

linkFromClipboard method

Pastes a link between an OLE object from the Clipboard and an OLE variable.

Syntax

linkFromClipboard ( ) Logical

Description

linkFromClipboard returns True if an OLE object is successfully pasted from the Clipboard and linked to an OLE variable; otherwise, it returns False.

After an OLE object is linked from the Clipboard, changes made while in Corel Paradox affect the underlying file. Compare this method to **readFromClipboard**, where changes made in Corel Paradox do not affect the underlying file.

Example

{button ,AL(`OPAL_TYPE_OLE;OPAL_METH_OLCANREADFROM;OPAL_METH_OLREADFROM;',0,"Defaultoverview",)} Related Topics

linkFromClipboard example

See the **canReadFromClipboard** example.

readFromClipboard method

Pastes an OLE object from the Clipboard into an OLE variable.

Syntax

readFromClipboard ( ) Logical

Description

readFromClipboard returns True if an OLE object is successfully pasted from the Clipboard into an OLE variable; otherwise, it returns False.

After an OLE object is pasted from the Clipboard, changes made while in Corel Paradox do not affect the underlying file. Compare this method to **linkFromClipboard**, where changes made in Corel Paradox affect the underlying file.

Example

{button ,AL(`OPAL_TYPE_OLE;OPAL_METH_OLCANREADFROM;',0,"Defaultoverview",)} Related Topics

readFromClipboard example

See the **canReadFromClipboard** example.

updateLinkNow method

Updates a linked OLE object.

Syntax

updateLinkNow ( ) Logical

Description

updateLinkNow updates a linked OLE object and returns True if successful. It returns False if the OLE object is an embedded object. You can use this method with **isLinked** to update the linked OLE fields in a table.

Example

{button ,AL(`OPAL_TYPE_OLE;OPAL_METH_OLISLINKED;',0,"Defaultoverview",)} **Related Topics**

updateLinkNow example

The following example scans the *Employee* table and updates any linked values in the OLE field named VoiceSample:

```
;btnUpdateLinks::pushButton
method pushButton(var eventInfo Event)
   var
      oleObj      OLE
      tcEmployee   TCursor
   endVar


   tcEmployee.open("employee")
   tcEmployee.edit()


   scan tcEmployee :
      oleObj = tcEmployee.VoiceSample    ; VoiceSample is an OLE field.
      if oleObj.isLinked() then
            oleObj.updateLinkNow()          ; Update the OLE variable.
            tcEmployee.VoiceSample = oleObj ; Assign the new value to the field in the
underlying table.
      endIf
   endScan


   tcEmployee.endEdit()
endMethod
```

writeToClipboard method

Copies an OLE variable to the Clipboard.

Syntax

writeToClipboard ( ) Logical

Description

writeToClipboard copies an original OLE object to the Clipboard. This method erases the Clipboard before copying the OLE object.

This method returns True if an OLE object is successfully copied to the Clipboard; otherwise, it returns False.

Example

{button ,AL(`OPAL_TYPE_OLE;OPAL_METH_OLREADFRO;',0,"Defaultoverview",)} **Related Topics**

writeToClipboard example

The following example reads an OLE field in a Corel Paradox table and assigns its value to an OLE variable. This code then writes the variable to the Clipboard, where it can be used by Corel Paradox or another application. The code assumes that EMPLOYEE.DB has an alpha field named Last Name and an OLE field named Picture.

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
  empTC TCursor
  oleImage OLE
endVar


empTC.open("Employee.db")     ; EMPLOYEE.DB has OLE images


if empTC.locate("Last Name", "Binkley") then


  oleImage = empTC.Picture    ; Picture is an OLE field
  oleImage.writeToClipboard() ; write contents of OLE field to variable


else
  msgStop("Error", "Can't find Binkley...")
endIf
endMethod
```

Point type

A Point variable contains information about a point on the screen. ObjectPAL considers the screen to be a two-dimensional grid, with the origin at the upper-left corner of the design object's container, the positive x values extending to the right, and the positive y values extending down. A Point has an x value and a y value, where *x* and *y* are measured in twips. A twip is 1/1440 of a logical inch, and 1/20 of a printer's point.

Methods defined for the Point type get and set information about screen coordinates and relative point positions. For example, a design object's size and position properties are specified in points.

ObjectPAL calculates point values relative to the container of the specified design object. This means that if a box contains a button, ObjectPAL calculates the button's position relative to the box. Similarly, if the button sits in an empty page, ObjectPAL calculates the button's position relative to the page. Methods that take or return Point values as arguments use this relative framework. You can use **convertPointWithRespectTo** defined for the UIObject type to convert values in different frameworks.

You can use Point operators (+, -, =, <, >, <=, and >=) to add, subtract, and compare Point variables. As the following example illustrates, these operators affect the x coordinates of each point first and then the y coordinates.

```
var
   p1, p2, p3 Point
endVar

   p1 = Point(10, 30)
   p2 = Point(10, 30)
   p3 = Point(10, 33)

   message(p1 + p2)  ; Displays (20, 60), because 10 + 10 = 20, and 30 + 30 = 60.
   message(p1 = p2)  ; Displays True. Both x and y coordinates are equal.
   message(p1 = p3)  ; Displays False. Both coordinates must be equal.
   message(p3 > p1)  ; Displays False. Both coordinates must be greater.
   message(p3 >= p1) ; Displays True. Both coordinates are either greater or equal.
```

The following table displays the methods for the Point type, including the derived methods from the AnyType type.

Methods for the Point type

AnyType	**Point**
blank	**distance**
dataType	**isAbove**
isAssigned	**isBelow**
isBlank	**isLeft**
isFixedType	**isRight**
view	**point**
	setX
	setXY
	setY
	x
	y

Print related ObjectPAL methods and examples

distance method

Returns the distance between two points, measured in twips.

Syntax

```
distance ( const pt Point ) Number
```

Description

distance returns the number of twips between a specified point and *pt*.

Example

{button ,AL(`OPAL_TYPE_POINT;OPAL_METH_PTISAB;OPAL_METH_PTISBE;OPAL_METH_PTISLE;OPAL_METH_PTISRI;OPAL_METH_PTX;OPAL_METH_PTY;',0,"Defaultoverview",)} Related Topics

distance example

The following example assumes a form contains 2 boxes: *redBox* and *brownBox*. The **pushButton** method for a button named *getDistance* determines the distance between the upper-left corners of the boxes:

```
; brownBox::pushButton
method pushButton(var eventInfo Event)
var
  p1, p2 Point
endVar
p1 = redBox.Position
p2 = brownBox.Position
msgInfo("Distance between boxes", p1.distance(p2))
; shows the distance between the top left corner of
; redBox and the top left corner of brownBox
endMethod
```

isAbove method

Reports whether a point is positioned above another point.

Syntax

```
isAbove ( const pt Point ) Logical
```

Description

isAbove returns True if the y coordinate of a point is less than the y coordinate of *pt*; otherwise, it returns False.

Example

{button ,AL(`OPAL_TYPE_POINT;OPAL_METH_PTISBE;OPAL_METH_PTISLE;OPAL_METH_PTISRI;',0,"Default overview",)} Related Topics

isAbove example

The following example uses the **pushButton** method for *convergeBoxes* to move *boxOne* closer to *boxTwo*, until the two boxes converge. Assume that *boxOne* is originally positioned above and left of *boxTwo*. Each time the button is clicked, *boxOne* moves down until it is on the same vertical plane and then moves to the right until it is covered by *boxTwo*.

```
; convergeBoxes::pushButton
method pushButton(var eventInfo Event)
var
  p1, p2 Point
endVar
p1 = boxOne.position             ; get the position of boxOne
p2 = boxTwo.position             ; get the position of boxTwo
if p1.isAbove(p2) then           ; compare the two points
  ; if p1 is higher than p2, move boxOne down
  boxOne.position = Point(p1.x(), p1.y() + 100)
else
  if p1.isLeft(p2) then
    ; if p1 is to the left of p2, move boxOne to the right
    boxOne.position = Point(p1.x() + 100, p1.y())
  endIf
endIf
endMethod
```

isBelow method

Reports whether a point is positioned below another point.

Syntax

```
isBelow ( const pt Point ) Logical
```

Description

isBelow returns True if the y coordinate of a point is greater than the y coordinate of *pt*; otherwise, it returns False.

Example

{button ,AL(`OPAL_TYPE_POINT;OPAL_METH_PTISAB;OPAL_METH_PTISLE;OPAL_METH_PTISRI;',0,"Defaultoverview",)} Related Topics

isBelow example

The following example uses the **pushButton** method for *convergeBoxes* to move *boxTwo* closer to *boxOne*, until the two boxes converge. Assume that *boxTwo* is originally positioned below and to the right of *boxOne*. Each time the button is clicked, *boxTwo* moves up until it is on the same vertical plane and then moves left until it is covered by *boxOne*.

```
; convergeBoxes::pushButton
method pushButton(var eventInfo Event)
var
  p1, p2 Point
endVar
p1 = boxOne.position             ; get the position of boxOne
p2 = boxTwo.position             ; get the position of boxTwo
if p2.isBelow(p1) then           ; compare the two points
  ; if p2 is lower than p1, move boxTwo up
  boxTwo.position = Point(p2.x(), p2.y() - 100)
else
  if p2.isRight(p1) then
    ; if p2 is to the left of p1, move boxTwo to the left
    boxTwo.position = Point(p2.x() - 100, p2.y())
  endIf
endIf
endMethod
```

isLeft method

Reports whether a point is positioned to the left of another point.

Syntax

```
isLeft ( const pt Point ) Logical
```

Description

isLeft returns True if the x coordinate of a point is less than the x coordinate of *pt*; otherwise, it returns False.

Example

{button ,AL(`OPAL_TYPE_POINT;OPAL_METH_PTISAB;OPAL_METH_PTISBE;OPAL_METH_PTISRI;',0,"Defaultoverview",)} **Related Topics**

isLeft example

See the **isAbove** example.

isRight method

Reports whether a point is positioned to the right of another point.

Syntax

```
isRight ( const pt Point ) Logical
```

Description

isRight returns True if the x coordinate of a point is greater than the x coordinate of *pt*; otherwise, it returns False.

Example

{button ,AL(`OPAL_TYPE_POINT;OPAL_METH_PTISAB;OPAL_METH_PTISBE;OPAL_METH_PTISLE;',0,"Defaultoverview",)} Related Topics

isRight example

See the **isBelow** example.

point procedure

Casts an expression as a Point.

Syntax

```
1. point ( const x LongInt, const y LongInt ) Point
2. point ( const newPoint Point ) Point
```

Description

point casts an expression as a Point.

Example

{button ,AL(`OPAL_TYPE_POINT;OPAL_METH_PTSETX;OPAL_METH_PTSETY;',0,"Defaultoverview",)}

Related Topics

point example

The following example varies the position of a box called *rateBox*. The values of an unbound field object named *rateField* range from 0 to 10. The position of *rateBox* is determined by the value in *rateField*. The following code is attached to the **changeValue** method for *rateField*:

```
; rateField::changeValue
method changeValue(var eventInfo ValueEvent)
Const
  baseXPosition = LongInt(3000)
  baseYPosition = LongInt(1000)
endConst
Var
  rateX   LongInt
endVar
try
  ; this if statement will fail if the field contents can't
  ; be compared to the integers 0 and 10 - for instance, if
  ; the user enters a string
  if eventInfo.newValue() >= 0 AND eventInfo.newValue() <= 10 then
    rateX = (eventInfo.newValue() * 400) + baseXPosition
    rateBox.Position = point(rateX, baseYPosition)
  else
    fail() ; if the value is a number but is out of range,
           ; call the fail block
  endIf
onFail
  disableDefault
  eventInfo.setErrorCode(CanNotDepart)
  msgStop("Stop", "Rating should be a number between 0 and 10.")
endTry


endMethod
```

setX method

Sets the x coordinate of a point.

Syntax

```
setX ( const newXValue LongInt )
```

Description

setX sets the x coordinate of a point to *newXValue*. If *newXValue* is not a LongInt, it is converted to a LongInt. This conversion may result in a loss of precision.

Example

{button ,AL(`OPAL_TYPE_POINT;OPAL_METH_PTSETY;',0,"Defaultoverview",)} Related Topics

setX example

In the following example, a form contains an ellipse named *circleOne* and a button named *moveRight*. The **pushButton** method for *moveRight* uses **setX** to change the horizontal coordinate of a point and then sets the position of *circleOne* to the changed point:

```
; moveRight::pushButton
method pushButton(var eventInfo Event)
var
  p1 Point
endVar
p1 = circleOne.position    ; get the position of the circle
p1.setX(p1.x() + 100)      ; add 100 twips to the x coordinate
circleOne.Position = p1    ; set the new position
message(p1)                ; display coordinates
endMethod
```

setXY method

Sets the x and y coordinates of a point.

Syntax

```
setXY ( const newXValue LongInt, const newYValue LongInt )
```

Description

setXY sets the x and y coordinates of a point to *newXValue* and *newYValue*. This method combines the functions of **setX** and **setY**. If *newXValue* and *newYValue* are not LongInts, they are converted to LongInts. This conversion may result in a loss of precision.

Example

{button ,AL(`OPAL_TYPE_POINT;OPAL_METH_PTSETX;OPAL_METH_PTSETY;',0,"Defaultoverview",)}

Related Topics

setXY example

In the following example, a form contains an ellipse called *circleOne* and a button named *moveDiagonal*. The **pushButton** method for *moveDiagonal* uses **setXY** to change the horizontal and vertical coordinates of a point and then sets the position of *circleOne* to the changed point:

```
; moveDiagonal::pushButton
method pushButton(var eventInfo Event)
var
  p1 Point
endVar
p1 = circleOne.position               ; get the position of the circle
p1.setXY(p1.x() + 100, p1.y() + 100)  ; add 100 twips to each coordinate
circleOne.Position = p1               ; set the new position
message(p1)                           ; display coordinates
endMethod
```

setY method

Sets the y coordinate of a point.

Syntax

```
setY ( const newYValue LongInt )
```

Description

setY sets the y coordinate of a point to *newYValue*. If *newYValue* is not a LongInt, it is converted to a LongInt, and precision may be lost.

Example

{button ,AL(`OPAL_TYPE_POINT;OPAL_METH_PTSETX;OPAL_METH_PTSETXY;',0,"Defaultoverview",)}

Related Topics

setY example

In the following example, a form contains an ellipse called *circleOne* and a button named *moveDown*. The **pushButton** method for *moveDown* uses **setY** to change the vertical coordinate of a point and then sets the position of *circleOne* to the changed point:

```
; moveDown::pushButton
method pushButton(var eventInfo Event)
var
  p1 Point
endVar
p1 = circleOne.position  ; get the position of the circle
p1.setY(p1.y() + 100)    ; add 100 twips to y coordinate
circleOne.Position = p1  ; set the new position
message(p1)              ; display coordinates
endMethod
```

x method

Returns the x coordinate of a point.

Syntax

x ( ) LongInt

Description

x returns the x coordinate of a point.

Example

{button ,AL(`OPAL_TYPE_POINT;OPAL_METH_PTSETX;OPAL_METH_PTSETY;OPAL_METH_PTY;',0,"Defaultoverview",)} Related Topics

x example

See the **setX** example.

y method

Returns the y coordinate of a point.

Syntax

y ( ) LongInt

Description

y returns the y coordinate of a point.

Example

{button ,AL(`OPAL_TYPE_POINT;OPAL_METH_PTSETY;OPAL_METH_PTSETX;OPAL_METH_PTX;',0,"Defaultoverview",)} Related Topics

y example

See the **setY** example.

PopUpMenu type

A PopUpMenu is a list of items that appears vertically in response to an Event (e.g., a mouse click). When the user chooses an item from a pop-up menu, the corresponding text is returned to the method. A PopUpMenu is distinct from a Menu, a list of items that appears horizontally in the application Menu Bar.

Choosing an item from a pop-up menu *does not* trigger the built-in **menuAction** method unless the pop-up menu is attached to a custom menu.

Using PopUpMenu methods, you can

- build a pop-up menu
- display the pop-up menu and return a selected item
- inspect the items in a pop-up menu
- provide keyboard access

The following table displays the methods for the PopUpMenu type, including several derived methods from the Menu type.

Methods for the PopUpMenu type

Menu	**PopUpMenu**
contains	**addArray**
count	**addBar**
empty	**addBreak**
remove	**addPopUp**
removeMen	**addSeparator**
	addStaticText
	addText
	show
	switchMenu

Print related ObjectPAL methods and examples

addArray method

Appends elements of an array to a pop-up menu.

Syntax

```
addArray ( const items Array[ ] String )
```

Description

addArray adds elements from an array to a pop-up menu.

Example

{button ,AL(`OPAL_TYPE_POPUPMENU;OPAL_METH_POABAR;OPAL_METH_MUABRE;OPAL_METH_POASEP; OPAL_METH_POASTE;OPAL_METH_POATEX;',0,"Defaultoverview",)} Related Topics

addArray example

In the following example, when the user right-clicks the field, a list of available payment types appears in a pop-up menu. The following code is attached to the **mouseRightUp** method for *paymentField*:

```
; paymentType::mouseRightUp
method mouseRightUp(var eventInfo MouseEvent)
var
  items  Array[4] String
  p1     PopUpMenu          ;  addArray is called for this PopUpMenu
  choice String
endVar


disableDefault              ; don't show default Font menu


items[1] = "Visa"
items[2] = "MasterCharge"
items[3] = "Check"
items[4] = "Cash"


p1.addArray(items)          ; add items array to the PopUpMenu
choice = p1.show()          ; display menu, remember choice
if not choice.isBlank() then
  self.value = choice
endIf


endMethod
```

addBar method

Adds a vertical bar to a pop-up menu.

Syntax

```
addBar ( )
```

Description

addBar adds a vertical bar to a pop-up menu. The **addBar** method creates a new column in the pop-up menu and inserts a vertical bar immediately before the new column. **addBar** is the vertical equivalent of **addSeparator**.

Example

{button ,AL(`OPAL_TYPE_POPUPMENU;OPAL_METH_POABRK;OPAL_METH_POASEP;',0,"Defaultoverview",)} **Related Topics**

addBar example

The following example displays a pop-up menu with two columns of choices. The first two choices are displayed in the left column, and all the remaining choices are displayed in the right column. This code is attached to a field's **mouseRightUp** method:

```
; navField::mouseRightUp
method mouseRightUp(var eventInfo MouseEvent)
var
  navPopUp   PopUpMenu    ; to show a navigate pop-up menu
  navChoice  String       ; store the menu choice
endVar

disableDefault                        ; don't show normal menu for field

navPopUp.addText("Previous record")   ; left menu
navPopUp.addText("First record")
navPopUp.addBar()                     ; add vertical bar
navPopUp.addText("Next record")       ; right menu
navPopUp.addText("Last record")

navChoice = navPopUp.show()           ; invoke menu
; ...
; process choice
; ...

endMethod
```

addBreak method

Starts a new column in a pop-up menu.

Syntax

```
addBreak ( )
```

Description

addBreak starts a new column in a pop-up menu. The first item added after the call to **addBreak** is displayed at the top of the column to the right of the previous column, and subsequent items follow below it. The **addBreak** method behaves like **addBar** in that it marks the beginning of a new column of choices. However, **addBreak** doesn't create a vertical bar between columns. **addBreak** doesn't create a cascading menu; use **addPopUp** instead.

 **Example**

{button ,AL(`OPAL_TYPE_POPUPMENU;OPAL_METH_POABAR;OPAL_METH_POASEP;',0,"Defaultoverview",)} **Related Topics**

addBreak example

The following example creates a pop-up menu with nine choices displayed in three vertical columns. This code is attached to *whereToButton*'s **pushButton** method:

```
; whereToButton::pushButton
method pushButton(var eventInfo Event)
var
  navPopUp      PopUpMenu       ; a pop-up of navigation choices
  navChoice     String          ; navigation chosen
endVar


navPopUp.addText("Home")        ; left menu
navPopUp.addText("Left")
navPopUp.addText("End")


navPopUp.addBreak()             ; start second column
navPopUp.addText("Up")
navPopUp.addText("Center")
navPopUp.addText("Down")


navPopUp.addBreak()             ; start third column
navPopUp.addText("PgUp")        ; right menu
navPopUp.addText("Right")
navPopUp.addText("PgDn")


navChoice = navPopUp.show()     ; invoke menu


; ... process choice


endMethod
```

addPopUp method

Adds a pop-up menu to the existing pop-up menu structure.

Syntax

```
addPopUp ( const menuName String, const cascadedPopup PopUpMenu )
```

Description

addPopUp adds *menuName* and *cascadedPopup* to the current pop-up menu structure, creating a cascading menu. *menuName* is displayed as an item in the original pop-up menu, and the first item in *cascadedPopup* appears next to it. Subsequent items in *cascadedPopUp* are displayed in a column below the first item.

Examples

{button ,AL(`OPAL_TYPE_POPUPMENU;OPAL_METH_POABRK;',0,"Defaultoverview",)} **Related Topics**

addPopUp method examples

Example1	Attaching a cascading menu to a Menu Bar item
Example2	Creating an unattached menu

addPopUp example 1

The following example uses **addPopUp** to attach a cascading menu to a Menu Bar item (a menu from the Menu type). In this example, the code attached to the built-in **open** method for *thisPage* creates and displays the pop-up menu structure. The code attached to *thisPage*'s **menuAction** handles the user's selection because the pop-up menus are attached to a Menu Bar item.

The following code is attached to the **open** method for *thisPage*:

```
; thisPage::open
method open(var eventInfo Event)
var
  mainMenu Menu
  subMenu1, subMenu2 PopUpMenu
endVar


  ; create 2nd level submenu
subMenu2.addText("&Time")
subMenu2.addText("&Date")


  ; add 2nd level to 1st level
subMenu1.addPopUp("&Utilities", subMenu2)


  ; add 1st level to Menu Bar
mainMenu.addPopUp("&File", subMenu1)


  ; display the Menu Bar
mainMenu.show()


endMethod
```

The following code is attached to *thisPage*'s **menuAction** method:

```
; thisPage::menuAction
method menuAction(var eventInfo MenuEvent)
var
  choice String
endVar


choice = eventInfo.menuChoice()
switch
  case choice = "&Time" : msgInfo("Current Time", time())
  case choice = "&Date" : msgInfo("Today's Date", date())
endSwitch


endMethod
```

addPopUp example 2

The following example uses **addPopUp** to create a cascading pop-up menu. This menu structure is not attached to a Menu Bar item, and the built-in **menuAction** method is not used. The code immediately following the call to **show** executes based on the user's selection.

The following code is attached to the **mouseRightUp** method for *pageTwo*:

```
; pageTwo::mouseRightUp
method mouseRightUp(var eventInfo MouseEvent)
var
  p1, p2, p3 PopUpMenu
  choice String
endVar


disableDefault                ; don't show normal pop-up menu


p2.addText("&Time")           ; build p2 and p3 submenus
p2.addText("&Date")
p3.addText("&Red")
p3.addText("&Green")
p3.addText("&Blue")


p1.addPopUp("&Utilities", p2) ; create Utilities item and attach p2 to it
p1.addPopUp("&Colors", p3)    ; create Colors item and attach p3 to it


choice = p1.show()            ; display menu and store selection to choice


switch                        ; now take action based on selection
  case choice = "&Red"   : self.color = Red
  case choice = "&Green" : self.color = Green
  case choice = "&Blue"  : self.color = Blue
  case choice = "&Time"  : msgInfo("Current Time", time())
  case choice = "&Date"  : msgInfo("Today's Date", date())
endSwitch


endMethod
```

addSeparator method

Adds a horizontal bar to a pop-up menu.

Syntax

```
addSeparator ( )
```

Description

addSeparator adds a horizontal bar to separate item groups in a pop-up menu. **addSeparator** is used to group similar commands within a menu.

Example

{button ,AL(`OPAL_TYPE_POPUPMENU;OPAL_METH_POABAR;OPAL_METH_POABRK;',0,"Defaultoverview" ,)} **Related Topics**

addSeparator example

The following example uses **addSeparator** to group pop-up menu commands. This code is attached to the built-in **open** method for *thisPage*:

```
; thisPage::open
method open(var eventInfo Event)
var
  mainMenu Menu
  subMenu1, clrMenu PopUpMenu
endVar


clrMenu.addText("&Red")
clrMenu.addText("&Blue")
clrMenu.addText("&White")


subMenu1.addText("&Time")
subMenu1.addText("&Date")
subMenu1.addSeparator()
subMenu1.addPopUp("&Page colors", clrMenu)
subMenu1.addSeparator()
subMenu1.addText("&About")


mainMenu.addPopUp("&Utilities", subMenu1)
mainMenu.show()
endMethod
```

The following code is attached to the built-in **menuAction** method for *thisPage*:

```
; thisPage::menuAction
method menuAction(var eventInfo MenuEvent)
var
  choice String
endVar
choice = eventInfo.menuChoice()
switch
  case choice = "&Red"   : self.color = Red
  case choice = "&Blue"  : self.color = Blue
  case choice = "&White" : self.color = White
  case choice = "&Time"  : msgInfo("Current Time", time())
  case choice = "&Date"  : msgInfo("Today's Date", date())
  case choice = "&About" : eventInfo.setId(MenuHelpAbout)
endSwitch
endMethod
```

addStaticText method

Adds a static (unselectable) text string to a pop-up menu.

Syntax

```
addStaticText ( const item String )
```

Description

addStaticText adds a static (unselectable) text string to a pop-up menu. Static text is used as the title (first item) in a pop-up menu.

Example

{button ,AL(`OPAL_TYPE_POPUPMENU;OPAL_METH_POATEX;',0,"Defaultoverview",)} Related Topics

addStaticText example

In the following example, when the user right-clicks the field, a list of available payment types is displayed in a pop-up menu. This example displays the first item as static text. The following code is attached to the **mouseRightUp** method for *paymentField*.

```
; paymentType::mouseRightUp
method mouseRightUp(var eventInfo MouseEvent)
var
  items  Array[4] String
  p1     PopUpMenu          ;  addArray is called for this PopUpMenu
  choice String
endVar


disableDefault              ; don't show default Font menu


items[1] = "Visa"
items[2] = "MasterCharge"
items[3] = "Check"
items[4] = "Cash"


                            ; display first item as static text
p1.addStaticText("Payment Method")
p1.addSeparator()           ; add a horizontal separator
p1.addArray(items)          ; add items array to the PopUpMenu
choice = p1.show()          ; display menu, remember choice
if not choice.isBlank() then
  self.value = choice
endIf


endMethod
```

addText method

Adds a selectable text string to a pop-up menu.

Syntax

```
1. addText ( const menuName String )
2. addText ( const menuName String, const attrib SmallInt )
3. addText ( const menuName String, const attrib SmallInt, const id SmallInt )
```

Description

addText adds a selectable text string to a pop-up menu. The pop-up menu can be displayed alone, or as part of a menu in the Menu Bar.

Syntax 1 uses *menuName* to specify the string to add to the pop-up menu.

Syntax 2, you can use *attrib* to preset the display attribute of *menuName*. ObjectPAL's MenuChoiceAttributes constants (e.g., MenuDisabled) for display attributes.

Syntax 3 is used only when the pop-up menu is attached to a Menu object. You can specify an *id* number (of type SmallInt) to identify the menu by number instead of by *menuName*. Then use in the built-in **menuAction** method, you use the *id* number to determine which menu the user chooses.

You can also use Syntax 3 to create a menu that provides the same functions as a built-in Corel Paradox menu. Use a MenuCommands constant to assign a value to the *id* argument. When the user chooses that item from a menu, Corel Paradox performs the default action. For example, the following line adds Next to the *puRecord* PopUpMenu and uses the MenuCommands constant MenuRecordNext to assign an ID value.

```
puRecord.addText("Next", MenuEnabled, MenuRecordNext)
```

You must display, enable, or disable menu items to ensure that the Corel Paradox operation that the user triggers is valid (e.g., you can only lock records in edit mode).

You can specify custom menu IDs, by adding a number or a user-defined menu constant to UserMenu. For example, the following code adds "File" to the *myPopup* PopUpMenu and specifies an *id* number for that menu item:

```
myPopup.addText("File", MenuEnabled, UserMenu + 1)
```

You can use an ampersand in an item so the user can select it using the keyboard. For example, the item &File would display as File, and the user could choose it by pressing F. When testing the user's choice, remember to include the ampersand. In this case, the returned value is &File, not File.

You can also use \t to insert a Tab between an item and its accelerator. For example, the item &Edit Data\tF9 displays Edit Data left-aligned and F9 right-aligned. In this case the string value returned is &Edit Data\tF9.

Examples

 Related Topics

addText examples

addText example 1

The following example displays a variation of the **addText** syntax.

For this example, assume a form has an unbound field named *payField*. When the user right-clicks the field, a list of available payment methods is displayed in a pop-up menu. The user can choose use the list to insert that value into the field or press ESC to cancel. The following code goes in the Var window for *payField*:

```
; payField::var
var
  payPopUp PopUpMenu
  mChoice  String
endVar
```

The following code is attached to the **open** method for *payField*. When the field opens for the first time, this code adds four items to the *payPopUp* PopUpMenu. This code prepares the pop-up menu for future display.

```
; payField::open
method open(var eventInfo Event)


payPopUp.addText("Visa")
payPopUp.addText("MasterCard")
payPopUp.addText("Check")
payPopUp.addText("Cash")


endMethod
```

The following code is attached to *payField*'s built-in **mouseRightUp** method. When the user right-clicks the field, this method uses **show** to display the menu and then inserts the user's choice in the unbound field.

```
; payField::mouseRightUp
method mouseRightUp(var eventInfo MouseEvent)


disableDefault               ; don't show default pop-up menu


mChoice = payPopUp.show()    ; display menu, store selection to mChoice
if not isBlank(mChoice) then ; if user does not press ESC
  self.value = mChoice       ; insert mChoice in unbound field
endIf
endMethod
```

addText example 2

The following example displays a variation of the **addText** syntax.

This example uses the *id* clause for pop-up menus attached to a Menu object. This code establishes user-defined constants to make it easy to remember the menu *id* assignments. The following code is added to Const window for *thisPage*.

```
; thisPage::const
Const
  kMenuRed   = 1  ; define constant values for menu ids
  kMenuBlue  = 2
  kMenuWhite = 3
  kMenuTime  = 4
  kMenuDate  = 5
  kMenuAbout = 6
endConst
```

The following code is attached to the **open** method for *thisPage*. To control the menu display attributes, this code uses built-in constants (e.g., MenuEnabled). To identify each menu item by number, the code uses the constants defined in the Const window for *thisPage* (*menuRed*, *menuBlue*, etc.).

```
; thisPage::open
method open(var eventInfo Event)
var
  mainMenu Menu
  subMenu1, clrMenu, puRecord PopUpMenu
endVar


  ; add text to pop-up menus and use user-defined constants
clrMenu.addText("&Red", MenuEnabled, kMenuRed + UserMenu)
clrMenu.addText("&Blue", MenuEnabled, kMenuBlue + UserMenu)
clrMenu.addText("&White", MenuEnabled, kMenuWhite + UserMenu)


subMenu1.addText("&Time", MenuEnabled, kMenuTime + UserMenu)
subMenu1.addText("&Date", MenuEnabled, kMenuDate + UserMenu)
subMenu1.addSeparator()
subMenu1.addPopUp("&Page colors", clrMenu)
subMenu1.addSeparator()
subMenu1.addText("&About", MenuEnabled, kMenuAbout + UserMenu)
; Build a pop-up menu to attach to the Record menu.
; Use ObjectPAL MenuCommands constants to assign item IDs.
puRecord.addText("&First", MenuEnabled, MenuRecordFirst)
puRecord.addText("&Prev", MenuEnabled, MenuRecordPrevious)
puRecord.addText("&Next", MenuEnabled, MenuRecordNext)
puRecord.addText("&Last", MenuEnabled, MenuRecordLast)
  ; attach pop-up menus to mainMenu and display the Menu Bar
mainMenu.addPopUp("&Utilities", subMenu1)
mainMenu.addPopUp("&Record", puRecord)
mainMenu.show()
endMethod
```

The following code is attached to the **menuAction** method for *thisPage*. This example evaluates menu selections by ID number:

```
; thisPage::menuAction
method menuAction(var eventInfo MenuEvent)
var
  menuId SmallInt
endVar


menuId = eventInfo.id()   ; store menu id number in menuId


switch
  case menuId = kMenuRed + UserMenu   : self.color = Red
  case menuId = kMenuBlue + UserMenu  : self.color = Blue
  case menuId = kMenuWhite + UserMenu : self.color = White
  case menuId = kMenuTime  + UserMenu : msgInfo("Time", time())
  case menuId = kMenuDate + UserMenu  : msgInfo("Date", date())
  case menuId = kMenuAbout + UserMenu : eventInfo.setId(MenuHelpAbout)
```

```
   ; No extra code is needed to handle choices from the Record menu,
   ; because item IDs were assigned using MenuCommands constants.
   ; Corel Paradox handles them automatically.
endSwitch


endMethod
```

show method

Displays a pop-up menu and returns the selected item.

Syntax

```
show ( [ const xTwips SmallInt, const yTwips SmallInt ] ) String
```

Description

show displays a pop-up menu and returns the selected item. If the user presses ESC without making a selection, the returned value is a zero-length string. The optional arguments *xTwips* and *yTwips* specify the coordinates of the upper left corner of the pop-up menu. If not specified, these arguments are set to the x and y coordinates of the pointer.

Example

{button ,AL(`OPAL_TYPE_POPUPMENU;OPAL_METH_MUEMPT;',0,"Defaultoverview",)} Related Topics

show example

For the following example, assume a form has an unbound field named *payField.* When the user right-clicks the field, a list of payment types is displayed in a pop-up menu. The user can choose from the list to insert that value into the field or press ESC to cancel. The following code is added to the Var window for *payField*:

```
; payField::var
var
  payPopUp PopUpMenu
  mChoice  String
endVar
```

The following code is attached to the **open** method for *payField*. When the field opens for the first time, this code adds four items to the *payPopUp* PopUpMenu. This code prepares the menu for future display.

```
; payField::open
method open(var eventInfo Event)


payPopUp.addText("Visa")
payPopUp.addText("MasterCard")
payPopUp.addText("Check")
payPopUp.addText("Cash")


endMethod
```

The following code is attached to *payField*'s built-in **mouseRightUp** method. When the user right-clicks the field, this method uses **show** to display the menu and inserts the user's choice in the unbound field.

```
; payField::mouseRightUp
method mouseRightUp(var eventInfo MouseEvent)


disableDefault               ; don't show default pop-up menu


mChoice = payPopUp.show()    ; display menu, store selection to mChoice
if not isBlank(mChoice) then ; if user does not press ESC
  self.value = mChoice       ; insert mChoice into unbound field
endIf
endMethod
```

switchMenu procedure

Builds and displays a pop-up menu, and handles the menu choice.

Syntax

```
switchMenu
   CaseList
   [ otherwise : Statements ]
endSwitchMenu
```

CaseList is any number of statements in the following form:

```
CASE menuItem : Statements
```

Description

switchMenu uses the values of the *menuItem* argument in each *CaseList* to create and display a pop-up menu. The *Statements* following each *menuItem* specify how to handle each menu choice. The optional otherwise clause specifies an action if the user closes the menu without making a choice (e.g., by pressing ESC).

Example

{button ,AL(`OPAL_TYPE_POPUPMENU;OPAL_METH_POATEX;OPAL_METH_POSHOW;',0,"Defaultoverview",)} Related Topics

switchMenu example

The following example uses **switchMenu** to create, display, and process a choice from a pop-up menu. A string describing the selection is displayed in the status line.

```
; actionButton::pushButton
method pushButton(var eventInfo Event)
switchMenu
  case "Add"    : message("Add selected.")
  case "Edit"   : message("Edit selected.")
  case "Delete" : message("Delete selected.")
  otherwise     : message("No selection from menu.")
endSwitchMenu
endMethod
```

Query type

An ObjectPAL Query variable is a query by example (QBE). You can use ObjectPAL to create and execute queries from methods in the same way that you use Corel Paradox interactively. You can execute a query from a query file, a query statement, or a string. Some queries require Corel Paradox to create temporary tables in your private directory.

Methods for the Query type

appendRow
appendTable
checkField
checkRow
clearCheck
createAuxTables
createQBEString
enumFieldStruct
executeQBE
getAnswerFieldOrder
getAnswerName
getAnswerSortOrder
getCheck
getCriteria
getQueryRestartOptions
getRowID
getRowNo
getRowOp
getTableID
getTableNo
hasCriteria
insertRow
insertTable
isAssigned
isCreateAuxTables
isEmpty
isExecuteQBELocal
isQueryValid
query
readFromFile
readFromString
removeCriteria
removeRow
removeTable
setAnswerFieldOrder
setAnswerName
setAnswerSortOrder
setCriteria
setLanguageDriver
setQueryRestartOptions
setRowOp
wantInMemoryTCursor
writeQBE

Print related ObjectPAL methods and examples

appendRow method

Appends a row to a query table image.

Syntax

```
appendRow ( const tableID SmallInt ) SmallInt
appendRow ( const tableName String ) SmallInt
```

Description

appendRow adds a new row to the specified table image in a query by example (QBE). The table is specified numerically by *tableID* or by *tableName*. **appendRow** returns the numeric *rowID* of the new row which is used to manipulate the row's contents. Even is rows are inserted or deleted ahead of the appended row, the *rowID* doesn't change.

Example

{button ,AL(`OPAL_TYPE_QUERY;OPAL_METH_QUERY;OPAL_METH_QUAPPENDTABLE;OPAL_METH_QUINSERTROW;OPAL_METH_QUINSERTTABLE;OPAL_METH_QUREMOVEROW;OPAL_METH_QUREMOVETABLE;OPAL_METH_QUSETROWOP;',0,"Defaultoverview",)} Related Topics

appendRow example

The following example appends a row to the query for the CUSTOMER.DB table:

```
method pushButton(var eventInfo Event)
var
   qVar Query
   rowID SmallInt
endVar


   qVar.appendTable( "CUSTOMER.DB" )
   rowID = qVar.appendRow( "CUSTOMER.DB" )
   qVar.setCriteria( "CUSTOMER.DB", rowID, "State/Prov", "CA or HI" )
   qvar.checkRow("Customer.db", rowID, CheckCheck)
   qvar.writeQBE("MyQBE")
endMethod
```

appendTable method

Appends a table to a query image.

Syntax

```
appendTable ( const tableName String ) SmallInt
```

Description

appendTable adds the table specified by *tableName* to the query image in a query by example (QBE) and returns a numeric ID which can be used to manipulate the table image's contents. Even if table images are inserted or deleted ahead of the appended table, its ID doesn't change.

Example

{button ,AL(`OPAL_TYPE_QUERY;OPAL_METH_QUERY;OPAL_METH_QUINSERTROW;OPAL_METH_QUINSERTTABLE;OPAL_METH_QUREMOVEROW;OPAL_METH_QUREMOVETABLE;OPAL_METH_QUSETROWOP;',0,"Defaultoverview",)} Related Topics

appendTable example

The following example appends a table to a query image:

```
method pushButton(var eventInfo Event)
var
   qVar Query
   rowID SmallInt
   tblID SmallInt
endVar


   tblID = qVar.appendTable( "CUSTOMER.DB" )
   rowID = qVar.appendRow( tblID )
   qVar.setCriteria( tblID , rowID, "State/Prov", "CA or HI" )
   qvar.checkRow("Customer.db", rowID, CheckCheck)
   qvar.writeQBE("MyQBE")
endMethod
```

checkField method

Creates a check mark in a specified field of a query table image.

Syntax

```
checkField ( const tableID SmallInt, const fieldID SmallInt, const checkType SmallInt ) Logical

checkField ( const tableID SmallInt, const fieldID SmallInt, const rowID SmallInt, const
checkType SmallInt ) Logical

checkField ( const tableID SmallInt, const fieldName String, const checkType SmallInt ) Logical

checkField ( const tableID SmallInt, const fieldName String, const rowID SmallInt, const
checkType SmallInt ) Logical

checkField ( const tableName String, const fieldID SmallInt, const checkType SmallInt ) Logical

checkField ( const tableName String, const fieldID SmallInt, const rowID SmallInt, const
checkType SmallInt ) Logical

checkField ( const tableName String, const fieldName String, const checkType SmallInt ) Logical

checkField ( const tableName String, const fieldName String, const rowID SmallInt, const
checkType SmallInt ) Logical
```

Description

checkField creates a check mark in a specified field. The table is specified numerically by *tableID* or by *tableName*. The field is specified by *fieldID* or *fieldName*. The corresponding row is specified by the row identifier *rowID*. If no row is specified, the **checkField** method defaults to the first row.

The ***checkType*** is one of the following qbeCheckType constants:

CheckCheck	Check mark (unique keys only)
CheckDesc	Descending order check
CheckGroup	GroupBy check
CheckNone	Invisible check
CheckPlus	Plus sign (include duplicate keys)

Example

{button ,AL(`OPAL_TYPE_QUERY;OPAL_METH_QUERY;OPAL_METH_QUCHECKROW;OPAL_METH_QUCLEARCHECK;OPAL_METH_QUGETCHECK;',0,"Defaultoverview",)} Related Topics

checkField example

The following example checks the State/Prov field in the CUSTOMER.DB table of the specified query image:

```
method pushButton(var eventInfo Event)
var
   qVar Query
   rowID SmallInt
   tblID SmallInt
   MyQBEValidateStr String
endVar


   tblID = qVar.appendTable( "CUSTOMER.DB" )
   rowID = qVar.appendRow( tblID )
   qVar.setCriteria( tblID , rowID, "State/Prov", "CA or HI" )
   qVar.checkField( tblID, rowID, "State/Prov", CheckPlus )
   MyQBEValidateStr = qVar.createQBEString()
   MyQBEValidateStr.view()
endMethod
```

checkRow method

Creates a check mark in each field of a specified row of a query table image.

Syntax

```
checkRow ( const tableName String, const rowID SmallInt, const checkType SmallInt ) Logical
checkRow ( const tableName String, const checkType SmallInt ) Logical
```

Description

checkRow creates a check mark in each field of a specified row of a table image. The table is specified numerically by *tableID* or by *tableName*. The row is specified by the row identifier *rowID*. If no row is specified, this method defaults to the first row.

The ***checkType*** is one of the following qbeCheckType constants:

CheckCheck	Check mark (unique keys only)
CheckDesc	Descending order check
CheckGroup	GroupBy check
CheckNone	Invisible check
CheckPlus	Plus sign (include duplicate keys)

Example

{button ,AL(`OPAL_TYPE_QUERY;OPAL_METH_QUERY;OPAL_METH_QUCHECKFIELD;OPAL_METH_QUCLEARCHECK;OPAL_METH_QUGETCHECK;',0,"Defaultoverview",)} **Related Topics**

checkRow example

The following example puts the CheckPlus symbol in every field in first row of the CUSTOMER.DB query table image and saves the query as ALLCust.QBE:

```
method pushButton(var eventInfo Event)
var
   qVar Query
endVar


   qVar.appendTable( "Customer.db" )
   qVar.checkRow( "Customer.db", CheckPlus )  ; row not specified,
                                              ; use first row
   qVar.writeQBE("ALLCust.QBE")
endMethod
```

clearCheck method

Deletes a check mark from a specified field or row of a query table image.

Syntax

```
clearCheck ( const tableID SmallInt, const fieldID SmallInt ) Logical
clearCheck ( const tableID SmallInt, const fieldName String ) Logical
clearCheck ( const tableID SmallInt, const rowID SmallInt, const fieldID SmallInt ) Logical
clearCheck ( const tableID SmallInt, const rowID SmallInt, const fieldName String ) Logical
clearCheck ( const tableName String, const fieldID SmallInt ) Logical
clearCheck ( const tableName String, const fieldName String ) Logical
clearCheck ( const tableName String, const rowID SmallInt, const fieldID SmallInt ) Logical
clearCheck ( const tableName String, const rowID SmallInt, const fieldName String ) Logical
```

Description

clearCheck removes a check mark from a specified field in the query by example (QBE). The table is specified numerically by *tableID* or by *tableName*. The field is specified by the *fieldID* or by *fieldName*. The row is specified by the row identifier *rowID*. If no row is specified, this method defaults to the first row.

Example

{button ,AL(`OPAL_TYPE_QUERY;OPAL_METH_QUERY;OPAL_METH_QUCHECKFIELD;OPAL_METH_QUCHECKROW;OPAL_METH_QUGETCHECK;',0,"Defaultoverview",)} Related Topics

clearCheck example

The following example removes the check mark from the State/Prov field in the CUSTOMER.DB query table image and then runs the query:

```
method pushButton(var eventInfo Event)
var
   qVar Query
endVar


   qVar.readFromFile( "monthly.qbe" )
   qVar.clearCheck( "Customer.db" , "State/Prov" )
   qVar.executeQBE()
endMethod
```

createAuxTables method

Enables the use of auxiliary tables.

Syntax

```
createAuxTables ( const useAuxTables Logical ) Logical
```

Description

createAuxTables enables the use of auxiliary tables if *useAuxTables* is set to True

Example

{button ,AL(`OPAL_TYPE_QUERY;OPAL_METH_QUERY;OPAL_METH_QUISCREATEAUXTABLES;',0,"Defaultoverview",)} **Related Topics**

createAuxTables example

The following example contains a query that uses auxiliary tables:

```
method pushButton(var eventInfo Event)var
myQBE Query
endvar

myQBE = Query

      Customer.db  |  Name        |
      Delete       |  Johnson..   |

endQuery

myQBE.createAuxTables(True)
myQBE.executeQBE()
endMethod
```

createQBEString method

Returns the QBE string of a query.

Syntax

createQBEString ( ) String

Description

createQBEString returns the QBE string of a query variable. If the query by example (QBE) is invalid, **createQBEString** returns a blank string and errorCode() determines the cause of the failure. The QBE **must** be a valid query against existing tables in order for this function to return a query string, so it should not be used to generate partial (incomplete) query strings or queries which will generate syntax errors if compiled or executed.

Example

{button ,AL(`OPAL_TYPE_QUERY;OPAL_METH_QUERY;OPAL_METH_QUWRITEQBE;OPAL_METH_QUISQUERYVALID;OPAL_METH_QUREADFROMFILE;OPAL_METH_QUREADFROMSTRING;',0,"Defaultoverview",)}

Related Topics

createQBEString example

The following example displays a QBE string from a modified version of the MONTHLY.QBE query.

```
method pushButton(var eventInfo Event)
var
   qVar Query
   qStr String
endVar


   qVar.readFromFile( "Monthly.qbe" )
   qVar.clearCheck( "Customer.db" , "State/Prov" )
   qVar.checkField( "Customer.db" , "Name", CheckPlus )
   qStr = qVar.createQBEString()
   if isblank( qStr ) then
      errorShow()
   else
      qStr.view( "Query String" )
   endif


endMethod
```

enumFieldStruct method

Lists the field structure of an answer table.

Syntax

```
1. enumFieldStruct ( const tableName String ) Logical
2. enumFieldStruct ( var inMemoryTC TCursor ) Logical
```

Description

enumFieldStruct lists the field structure of the answer table that is generated from the query by example (QBE) statement. Syntax 1 creates a Corel Paradox table, and Syntax 2 stores the information in a TCursor variable. **enumFieldStruct** returns True if successful; otherwise, it returns False.

Syntax 1 creates the Corel Paradox table specified in *tableName*. If *tableName* exists, **enumFieldStruct** overwrites it without confirmation. You can include an alias or path in *tableName* but if no alias or path exists, Corel Paradox creates *tableName* in the working directory.

Syntax 2 stores the information in the TCursor variable named *inMemoryTC.* You pass *inMemoryTC* as an argument.

The following table describes the structure of the table (Syntax 1) or TCursor (Syntax 2):

Field	Type	Description
Field Name	A31	Name of field
Type	A31	Data type of field
Size	S	Size of field
Dec	S	Number of decimal places in the field (0 if field type doesn't support decimal places)
Key	A1	* = key field, blank = not key field
_Required Value	A1	T = required field, N (or blank) = Not required
_Min Value	A255	Minimum value, if specified; otherwise blank
_Max Value	A255	Maximum value, if specified; otherwise blank
_Default Value	A255	Default value, if specified; otherwise blank
_Picture Value	A175	Picture, if specified; otherwise blank
_Table Lookup	A255	Name of lookup table; including full path if the lookup table is not in :WORK:
_Table Lookup Type	A1	Type of lookup table. 0 (or blank) = no lookup table, 1 = Corel Paradox
_Invariant Field ID	S	Ordinal position of field in the table (first field = 1, second field = 2, etc.)

Example

enumFieldStruct example

The following example creates the Corel Paradox table MYANSWER.DB containing the structure of the answer table that is built by the query MYQUERY.QBE:

```
method pushButton(var eventInfo Event)
var
   qVar      Query
endVar
   qVar.readFromFile( "myquery.qbe" )
   qVar.enumFieldStruct("QSTRUCT.DB")
endMethod
```

executeQBE method/procedure

Executes a query by example (QBE).

Syntax

```
1. (Method)
executeQBE ( [ { const ansTbl String |
                   var ansTbl Table |
                 var ansTbl TCursor } ] ) Logical

2. (Procedure)
executeQBE ( var db Database, var qVar Query
                [ , { const ansTbl String |
                         var ansTbl Table |
                       var ansTbl TCursor } ] ) Logical
```

Description

executeQBE executes the query assigned to a Query variable and writes the results to :PRIV:ANSWER.DB or to the table specified in *ansTbl*. You can assign a query to a Query variable using a **query** statement. Create a **query** statement by calling **readfromFile** or **readfromString** or by building it with **appendTable**, **appendRow**, or **setCriteria**.

Syntax 1 calls **executeQBE** as a method. You can write the query result to *ansTbl* where *ansTbl* is a table name, a Table variable, or a TCursor. If *ansTbl* is not specified, **executeQBE** writes the results to ANSWER.DB in the private directory.

Syntax 2 calls **executeQBE** as a procedure. Specify a Database variable in *db* and a Query variable in *qVar*. You can write the query result to *ansTbl* where *ansTbl* is a table name, a Table variable, or a TCursor. If *ansTbl* is not specified, **executeQBE** writes the results to ANSWER.DB in the private directory.

The following notes apply to both syntaxes:

- If you specify the table name as a string and don't include a file extension, *ansTbl* defaults to specify a Corel Paradox.
- If you specify *ansTbl* as a Table variable, *ansTbl* must be assigned and valid.
- If you specify *ansTbl* as a TCursor, the results are stored in memory only.
- If **executeQBE** is successful (*ansTbl* or ANSWER.DB is created)this method returns True; otherwise it returns False. **executeQBE** returns True even if the resulting table is empty.

Examples

{button ,AL(`OPAL_TYPE_QUERY;OPAL_METH_QUERY;OPAL_METH_QUWRITEQBE;OPAL_METH_QUREADFROMFILE;OPAL_METH_QUREADFROMSTRING;',0,"Defaultoverview",)} Related Topics

executeQBE method examples

Example1 **executeQBE** as a method

Example2 **executeQBE** as a procedure

executeQBE example 1

The following example calls **executeQBE** as a method. The **pushButton** method for the *getReceivables* button constructs a query statement, assigns it to a Query variable and then runs it with **executeQBE**. The query statement in this example is an insert query: it retrieves records from CUSTOMER.DB and ORDERS.DB and inserts them into the existing *MyCust* table. The selection criteria for this example uses a tilde variable *myState* that includes Oregon customers in the results. Because OR is an ObjectPAL keyword, the *myState* variable must evaluate to a quoted string to distinguish it from the abbreviation for Oregon.

```
method pushButton(var eventInfo Event)
var
   qVar     Query
   myState  String
   tv       TableView
endVar


; add samp alias for the sample directory
addAlias("samp", "Standard", "c:\\Corel\\Suite8\\Paradox\\samples")


; OR is the abbreviation for Oregon, but because it's
; also an ObjectPAL keyword, it must be enclosed in quotes.
myState = "\"OR\""


; now use myState as a tilde variable in this query statement
qVar = Query


      :samp:Customer.db|Customer No|Name |State/Prov|Phone  |
                       |_cust      |_name|  ~myState|_phone |


      :samp:Orders.db |Customer No|Balance Due|
                      |_cust      |0, _balDue |


           myCust.db |Customer No|Name |Balance Due|Phone |
             insert  |_cust      |_name|_balDue    |_phone|


      EndQuery


qVar.executeQBE("myCust.db")   ; put results into myCust.db
tv.open("myCust.db")           ; view the table


endMethod
```

executeQBE example 2

The following example calls **executeQBE** as a procedure. The **pushButton** method for the *getReceivables* button constructs a query statement, assigns it to a Query variable and then runs it with **executeQBE**. The query statement in this example is an insert query: it retrieves records from CUSTOMER.DB and ORDERS.DB and inserts them into the existing *MyCust* table. The selection criteria for this example uses a tilde variable *myState* that includes Oregon customers in the results. Because OR is an ObjectPAL keyword, the *myState* variable must evaluate to a quoted string to distinguish it from the abbreviation for Oregon.

```
method pushButton(var eventInfo Event)
var
   db      Database
   qVar    Query
   myState String
   tv      TableView
endVar


db.open() ; Get a handle to the default database.


; add samp alias for the sample directory
addAlias("samp", "Standard", "c:\\Corel\\Suite8\\Paradox\\samples")


; OR is the abbreviation for Oregon, but because it is also
; a Corel Paradox keyword it must be enclosed in quotes
myState = "\"OR\""


; now use myState as a tilde variable in this query statement
qVar = Query

       :samp:Customer.db|Customer No|Name |State/Prov|Phone  |
                        |_cust      |_name|  ~myState|_phone |

        :samp:Orders.db |Customer No|Balance Due|
                        |_cust      |0, _balDue |

              myCust.db |Customer No|Name |Balance Due|Phone |
                insert  |_cust      |_name|_balDue    |_phone|

        EndQuery

executeQBE(db, qVar, "myCust.db")   ; put results into myCust.db
tv.open("myCust.db")                ; view the table

endMethod
```

getAnswerFieldOrder method

Retrieves the field names of the custom field order in the answer table generated by a query.

Syntax

```
getAnswerFieldOrder ( var fieldOrder Array[] String ) Logical
```

Description

getAnswerFieldOrder retrieves an array of the fields in the answer table for the current query, when a custom field order is specified. If a custom field order is not specified for the query, this method returns an empty array. If the query compiles successfully, the array *fieldOrder* is filled with the field names in the answer table. These names can rearranged and the array can be submitted to the **setAnswerFieldOrder** and **setAnswerSortOrder** methods. The array must be resizable.

Example

{button ,AL(`OPAL_TYPE_QUERY;OPAL_METH_QUGETANSWERNAME;OPAL_METH_QUGETANSWERSORTORDER;OPAL_METH_QUERY;OPAL_METH_QUSETANSWERFIELDORDER;OPAL_METH_QUSETANSWERNAME;OPAL_METH_QUSETANSWERSORTORDER;',0,"Defaultoverview",)} Related Topics

getAnswerFieldOrder example

The following example retrieves the existing field order specified in MYQUERY.QBE, reorders the fields, and then uses **setAnswerFieldOrder** to put the new order in place.

```
method pushButton(var eventInfo Event)
var
   qVar  Query
   arFields  Array[] String
endVar
qVar.readFromFile( "myquery.qbe" )
qVar.getAnswerFieldOrder( arFields )
   if arFields.size() > 0 then   ; swap the first and third fields
                                 ; in the answer table.
   arFields.exchange(1,3)
   qVar.setAnswerFieldOrder( arFields )
   qVar.executeQBE()
   endif
endMethod
```

getAnswerName method

Retrieves the name of the answer table.

Syntax

```
getAnswerName () String
```

Description

getAnswerName retrieves the name of the answer table that is produced by the query.

Example

{button ,AL(`OPAL_TYPE_QUERY;OPAL_METH_QUGETANSWERFIELDORDER;OPAL_METH_QUGETANSWERSORTORDER;OPAL_METH_QUERY;OPAL_METH_QUSETANSWERFIELDORDER;OPAL_METH_QUSETANSWERNAME;OPAL_METH_QUSETANSWERSORTORDER;',0,"Defaultoverview",)} Related Topics

getAnswerName example

The following example allows the user to change the answer table name for MYQUERY.QBE:

```
method pushButton(var eventInfo Event)
var
   qVar  Query
   AnsTblName String
endVar
if msgQuestion("Query",
               "Would you like to change the "
             + "answer table name?") = "Yes" then
   qVar.readFromFile("MYQUERY.QBE")
   AnsTblName = qVar.getAnswerName()
   AnsTblName.view("Make changes below")
   qVar.setAnswerName(AnsTblName)
   qVar.writeQBE("MYQUERY.QBE")
endif
endMethod
```

getAnswerSortOrder method

Retrieves the custom sort order specified for the answer table.

Syntax

```
getAnswerSortOrder ( var sortFields Array[] String ) Logical
```

Description

getAnswerSortOrder retrieves the custom sort order specified for the answer table. If a custom sort order is not specified, this method returns an empty array. The array *sortFields* contains an ordered list of field names. After you retrieve an array of these field names using **getAnswerSortOrder**, you can change the sort order.

If you retrieve the list of fields and then change the answer field list (e.g., by unchecking a field), the array is instantly outdated. You must remove the modified field from your array before attempting to use this array for field sorting.

Example

{button ,AL(`OPAL_TYPE_QUERY;OPAL_METH_QUGETANSWERFIELDORDER;OPAL_METH_QUGETANSWERNAME;OPAL_METH_QUERY;OPAL_METH_QUSETANSWERFIELDORDER;OPAL_METH_QUSETANSWERNAME;OPAL_METH_QUSETANSWERSORTORDER;',0,"Defaultoverview",)} Related Topics

getAnswerSortOrder example

The following example retrieves the field list from MYQUERY.QBE, reorders the fields, and saves the new sort order back into the query using **setAnswerSortOrder**:

```
method pushButton(var eventInfo Event)
var
   qVar  Query
   arFields  Array[] String
endVar
qVar.readFromFile( "myquery.qbe" )
qVar.getAnswerSortOrder( arFields )
   if arFields.size() > 0 then   ; swap the first and third fields
                                 ; in the sort order.
      arFields.exchange(1,3)
      qVar.setAnswerSortOrder( arFields )
      qVar.executeQBE()
   endif
endMethod
```

getCheck method

Returns the check type for a specified field in a query image.

Syntax

```
getCheck ( const tableID SmallInt, const fieldID SmallInt ) SmallInt
getCheck ( const tableID SmallInt, const fieldName String ) SmallInt
getCheck ( const tableID SmallInt, rowID SmallInt, const fieldID SmallInt ) SmallInt
getCheck ( const tableID SmallInt, rowID SmallInt, const fieldName String ) SmallInt
getCheck ( const tableName String, const fieldID SmallInt ) SmallInt
getCheck ( const tableName String, const fieldName String ) SmallInt
getCheck ( const tableName String, rowID SmallInt, const fieldID SmallInt ) SmallInt
getCheck ( const tableName String, rowID SmallInt, const fieldName String ) SmallInt
```

Description

getCheck returns the check type for a specified field in a query image. The table is specified numerically by *tableID* or by *tableName*. The field is specified by the *fieldID* or by *fieldName*. The row is specified by the row identifier *rowID*. If no row is specified, this method defaults to the first row.

The ***checkType*** is one of the following qbeCheckType constants:

CheckCheck	Check mark (unique keys only)
CheckDesc	Descending order check
CheckGroup	GroupBy check
CheckNone	Invisible check
CheckPlus	Plus sign (include duplicate keys)

Example

{button ,AL(`OPAL_TYPE_QUERY;OPAL_METH_QUERY;OPAL_METH_QUCHECKFIELD;OPAL_METH_QUCHECKROW;OPAL_METH_QUCLEARCHECK;',0,"Defaultoverview",)} Related Topics

getCheck example

The following example changes the type of check used in the State/Prov field of the CUSTOMER.DB query table image from CheckPlus to CheckDesc.

```
method pushButton(var eventInfo Event)
var
   qVar Query
   qStr String
endVar
qVar.readFromFile( "monthly.qbe" )
   if qVar.getCheck( "Customer.db" , "State/Prov" ) = CheckPlus then
      qVar.CheckField( "Customer.db" , "State/Prov" , CheckDesc )
      qVar.writeQBE("Monthly.QBE")
   endif
endMethod
```

getCriteria method

Returns the query expression used in a query image.

Syntax

```
getCriteria ( const tableID SmallInt, const fieldID SmallInt ) String
getCriteria ( const tableID SmallInt, const fieldName String ) String
getCriteria ( const tableID SmallInt, const rowID SmallInt, const fieldID SmallInt ) String
getCriteria ( const tableID SmallInt, const rowID SmallInt, const fieldName String ) String
getCriteria ( const tableName String, const fieldID SmallInt ) String
getCriteria ( const tableName String, const fieldName String ) String
getCriteria ( const tableName String, const rowID SmallInt, const fieldID SmallInt ) String
getCriteria ( const tableName String, const rowID SmallInt, const fieldName String ) String
```

Description

getCriteria returns the selection conditions and calculation statements in a specified field of a query image. The table is specified numerically by *tableID* or by *tableName*. The field is specified by the *fieldID* or by *fieldName*. The row is specified by the row identifier *rowID*. If no row is specified, this method defaults to the first row.

This expression does not include the check mark, but does contain the remaining field contents.

Example

{button ,AL(`OPAL_TYPE_QUERY;OPAL_METH_QUERY;OPAL_METH_QUHASCRITERIA;OPAL_METH_QUISQUERYVALID;OPAL_METH_QUREMOVECRITERIA;OPAL_METH_QUSETCRITERIA;',0,"Defaultoverview",)}

Related Topics

getCriteria example

The following example changes the selection conditions for the Name field in the CUSTOMER.DB query table image:

```
method pushButton(var eventInfo Event)
var
   qVar Query
   NameCriteria String
endVar
   qVar.readFromFile( "monthly.qbe" )
   NameCriteria = qVar.getCriteria ( "Customer.db" , "Name")
                                   ; default to the first row
   NameCriteria = NameCriteria + " or Unisco"
   qVar.setCriteria( "Customer.db" , "Name" , NameCriteria )


endMethod
```

getQueryRestartOptions method

Returns a value representing the user's query restart setting.

Syntax

getQueryRestartOptions ( ) SmallInt

Description

getQueryRestartOptions returns an integer value representing the user's query restart setting. Use one of the following ObjectPAL QueryRestartOptions constants to test the value:

QueryDefault	Use the options specified interactively by using the Query Restart Options dialog box.
QueryLock	Lock other users out of the tables needed to run the query. If Corel Paradox cannot lock a table, it does not run the query.
QueryNoLock	Continue to run the query if a change is made to the data during its execution.
QueryRestart	Restart the query. Specify QueryRestart to get a snapshot of the data as it existed at a particular instant.

Example

{button ,AL(`OPAL_TYPE_QUERY;OPAL_TYPE_SQL;OPAL_METH_QUSETQRESTART;',0,"Defaultoverview",)}
Related Topics

getQueryRestartOptions example

See the **setQueryRestartOptions** example.

getRowID method

Returns the row identifier for a specified sequence row number.

Syntax

```
getRowID ( const tableID SmallInt, const seqNo SmallInt ) SmallInt
```

Description

getRowID returns the *rowID* for the specified sequence. The *rowID* is any number, regardless of where the row resides in the table image on the query workspace. The table is specified numerically by *tableID*.

To determine the *rowID* of a specified sequence you must convert the row's sequence number to the *rowID*. For example, the second row of the Customer.db table image might have a *rowID* of 32760.

Example

{button ,AL(`OPAL_TYPE_QUERY;OPAL_METH_QUERY;OPAL_METH_QUGETROWNO;OPAL_METH_QUGETTABLEID;OPAL_METH_QUGETTABLENO;',0,"Defaultoverview",)} Related Topics

getRowID example

The following example returns the row identifier of the second row in CUSTOMER.DB, assigns the name secondRow, changes the criteria of the Country field, and runs the query:

```
method pushButton(var eventInfo Event)
var
   qVar Query
   secondRow SmallInt
endVar
   qVar.readFromFile( "monthly.qbe" )
   secondRow = qVar.getRowID( qvar.getTableID(1), 2 )
   qVar.setCriteria( "Customer.db", secondRow, "Country", "Fiji" )
   qVar.executeQBE()
endMethod
```

getRowNo method

Returns the sequence number of a specific row.

Syntax

```
getRowNo ( const tableID SmallInt, const rowID SmallInt ) SmallInt
```

Description

getRowNo returns the sequence number of the row specified by *rowID*. The sequence number is the complement of **getRowID**. Given a unique numeric row identifier (*rowID*), **getRowNo** returns the current position (sequence) of the row in the query table image. For example, a *rowID* of 32760 might be the third row in a table image.

Example

{button ,AL(`OPAL_TYPE_QUERY;OPAL_METH_QUERY;OPAL_METH_QUGETROWID;OPAL_METH_QUGETTABLEID;OPAL_METH_QUGETTABLENO;',0,"Defaultoverview",)} Related Topics

getRowNo example

The following example creates a Query variable and appends the CUSTOMER.DB table image to the query. The code then prints the new row's sequence number.

```
method pushButton(var eventInfo Event)
var
   qVar         Query
   seqNo,
   rowID,
   tableID      SmallInt
endVar
   tableID=qVar.appendTable( "Customer.db" )
   rowID = qVar.appendRow( "Customer.db" )
   seqNo = qVar.getRowNo( tableID,rowID )
   message( "The newly appended row is row number ",seqNo,
            " in the customer.db query image" )
endMethod
```

getRowOp method

Retrieves the row operator set for a specified row.

Syntax

```
getRowOp ( const tableID SmallInt [, const rowID SmallInt] ) SmallInt
getRowOp ( const tableName String [, const rowID SmallInt] ) SmallInt
```

Description

getRowOp returns the row operator set for a specified row. The table is specified numerically by *tableID* or by *tableName*. The row can be specified by the row identifier *rowID*. If no row is specified, this method defaults to the first row.

The ***rowOperator*** is one of the following values:

qbeRowDelete	Delete operator
qbeRowInsert	Insert operator
qbeRowNone	No operator
qbeRowSet	Set operator

Example

{button ,AL(`OPAL_TYPE_QUERY;OPAL_METH_QUERY;OPAL_METH_QUAPPENDTABLE;OPAL_METH_QUINSERTROW;OPAL_METH_QUINSERTTABLE;OPAL_METH_QUREMOVEROW;OPAL_METH_QUREMOVETABLE;OPAL_METH_QUSETROWOP;',0,"Defaultoverview",)} Related Topics

getRowOp example

The following example deletes records containing blank Customer No fields:

```
method pushButton(var eventInfo Event)
var
   qVar  Query
   rowop  SmallInt
endVar


   qVar.readFromFile( "Sample.qbe" )
   rowop = qVar.getRowOp( "Customer.db" )


endMethod
```

getTableID method

Returns the tableID for a specified table in the query image.

Syntax

```
getTableID ( const seqNo SmallInt ) SmallInt
```

Description

getTableID returns the tableID for a specified table in the query image. This ID differs from the table's sequential number, and using the sequential number results in errors. You can replace the tableID with the table name in those methods that accept the table name as a valid entry.

Example

{button ,AL(`OPAL_TYPE_QUERY;OPAL_METH_QUERY;OPAL_METH_QUGETROWID;OPAL_METH_QUGETROWNO;OPAL_METH_QUGETTABLENO;',0,"Defaultoverview",)} Related Topics

getTableID example

The following example retrieves the table ID for the third table and the row ID for the second row of the query MONTHLY.QBE. The code then uses these IDs to determine the criteria set in the Name field.

```
method pushButton(var eventInfo Event)
var
   qVar Query
   thirdTableID, secondrowID  SmallInt
   condition String
endVar


qVar.readFromFile("MONTHLY.QBE")
thirdTableID = qVar.getTableID(3)
secondRowID = qVar.getRowID(thirdTableID, 2)
condition = qVar.getCriteria(thirdTableID, secondRowID, "Name")
msgInfo("Condition", "The criteria for the Name field in the "
      + "second row of the third table is " + condition)
endMethod
```

getTableNo method

Returns the table number for a specified table.

Syntax

```
getTableNo ( const tableID SmallInt ) SmallInt
```

Description

getTableNo returns the table number for the table specified by *tableID*. Given a unique numeric *tableID*, **getTableNo** returns its current position in the query. For example, if a *tableID* of 32760 represents the second table on the query workspace, this method returns a value of 2.

Example

{button ,AL(`OPAL_TYPE_QUERY;OPAL_METH_QUERY;OPAL_METH_QUGETROWID;OPAL_METH_QUGETROWNO;OPAL_METH_QUGETTABLEID;',0,"Defaultoverview",)} Related Topics

getTableNo example

The following example displays a specified table's position in the query:

```
method pushButton(var eventInfo Event)
var
   qVar Query
   qStr String
   seqNo, rowID, newTableID   SmallInt
endVar
   qVar.readFromFile( "monthly.qbe" )
   newTableID = qVar.appendTable( "Vendors.db" )
   seqNo = qVar.getTableNo( newTableID )
   message( "The newly appended table is table number ",
            seqNo," in the query image" )
endMethod
```

hasCriteria method

Indicates whether a specific field contains query criteria:

Syntax

```
hasCriteria ( const tableID SmallInt, const fieldID SmallInt ) Logical
hasCriteria ( const tableID SmallInt, const fieldName String ) Logical
hasCriteria ( const tableID SmallInt, const rowID SmallInt, const fieldID SmallInt ) Logical
hasCriteria ( const tableID SmallInt, const rowID SmallInt, const fieldName String ) Logical
hasCriteria ( const tableName String, const fieldID SmallInt ) Logical
hasCriteria ( const tableName String, const fieldName String ) Logical
hasCriteria ( const tableName String, const rowID SmallInt, const fieldID SmallInt ) Logical
hasCriteria ( const tableName String, const rowID SmallInt, const fieldName String ) Logical
```

Description

hasCriteria returns a Logical value indicating whether a specified field contains query criteria. The table is specified numerically by *tableID* or by *tableName*. The field is specified by the *fieldID* or by *fieldName*. The row is specified by the row identifier *rowID*, or omitted to default to the first row.

hasCriteria examines the field for a query expression but does **not** detect check marks. Use **getCheck** to determine whether a field is checked.

Example

{button ,AL(`OPAL_TYPE_QUERY;OPAL_METH_QUERY;OPAL_METH_QUGETCHECK;OPAL_METH_QUGETCRITERIA;OPAL_METH_QUISQUERYVALID;OPAL_METH_QUREMOVECRITERIA;OPAL_METH_QUSETCRITERIA;',0,"Defaultoverview",)} **Related Topics**

hasCriteria example

The following example retrieves criteria from the Sale Date field in the table ORDERS.DB in the query and runs the query:

```
method pushButton(var eventInfo Event)
var
   qVar          Query
   newTableID    SmallInt
   DateCriteria  String
endVar
   qVar.readFromFile( "monthly.qbe" )
   if qVar.hasCriteria( "Orders.db" , "Sale Date" ) then
      DateCriteria = qVar.getCriteria( "Orders.db" , "Sale Date")
   else
      DateCriteria = ""
   endif
   DateCriteria.view( "Enter Date Criteria" )
   qVar.setCriteria( "Orders.db", "Sale Date", DateCriteria )
   qVar.executeQBE()
endMethod
```

insertRow method

Adds a new row above an existing row in the query.

Syntax

```
insertRow ( const tableID SmallInt, beforeRowID SmallInt ) SmallInt
insertRow ( const tableName String, beforeRowID SmallInt ) SmallInt
```

Description

insertRow adds a new row above an existing row in the query. The table is specified numerically by *tableID* or by *tableName*. The parameter *beforeRowID* specifies the ID of the row which should be pushed down by the new row.

insertRow returns a SmallInt representing the row ID of the new row.

Example

{button ,AL(`OPAL_TYPE_QUERY;OPAL_METH_QUERY;OPAL_METH_QUAPPENDTABLE;OPAL_METH_QUINSERTTABLE;OPAL_METH_QUREMOVEROW;OPAL_METH_QUREMOVETABLE;OPAL_METH_QUSETROWOP;',0,"Defaultoverview",)} Related Topics

insertRow example

The following example creates a query, based on the CUSTOMER.DB table, that retrieves customer records for two cities. After one row is appended and its query criteria set, another row is inserted and its criteria is set.

```
method pushButton(var eventInfo Event)
var
   qVar                 Query
   firstRow, secondRow  SmallInt
endVar


   qVar.appendTable( "CUSTOMER.DB" )
   secondRow = qVar.appendRow( "CUSTOMER.DB " )
   qVar.checkRow( "CUSTOMER.DB" , CheckCheck )
   qVar.setCriteria( "CUSTOMER.DB", "City", "Waterville")
   qVar.setCriteria( "CUSTOMER.DB", "Country", "USA")
   firstRow = qVar.insertRow( "CUSTOMER.DB", secondRow)
   qvar.checkRow( "CUSTOMER.DB", CheckCheck)
   qVar.setCriteria( "CUSTOMER.DB", "City", "Vancouver")
   qVar.setCriteria( "CUSTOMER.DB", "Country", "Canada")
   qVar.writeQBE( "TwoCity.QBE" )
endMethod
```

insertTable method

Inserts a new table above an existing table in the query.

Syntax

```
insertTable ( const tableName String, const beforeTableID SmallInt ) SmallInt
insertTable ( const tableName String, const beforeTableName String ) SmallInt
```

Description

insertTable inserts a new table above an existing table in the query and returns the *tableID* for the new table. The parameter *tableName* specifies the name of the table to insert. The parameters *beforeTableID* and *beforeTableName* specify the ID and name (respectively) of the table that follows the new table.

Example

{button ,AL(`OPAL_TYPE_QUERY;OPAL_METH_QUERY;OPAL_METH_QUAPPENDTABLE;OPAL_METH_QUINSERTROW;OPAL_METH_QUREMOVEROW;OPAL_METH_QUREMOVETABLE;OPAL_METH_QUSETROWOP;',0,"Defaultoverview",)} Related Topics

insertTable example

The following example creates a query that includes the Customer and Orders tables. The two tables are linked with an example element on their common field, Customer No, and all fields are checked in the Customer table. The query produces an answer table that lists all customer records containing order records.

```
method pushButton(var eventInfo Event)
var
   qVar  Query
endVar
qVar.appendTable("CUSTOMER.DB")
qVar.checkRow("CUSTOMER.DB", CheckCheck)
qVar.setCriteria("CUSTOMER.DB", "Customer No", "_Join1")
qVar.insertTable("ORDERS.DB", "CUSTOMER.DB")
qVar.setCriteria("CUSTOMER.DB", "Customer No", "_Join1")
qVar.executeQBE()
endMethod
```

isAssigned method

Reports whether a Query variable has an assigned value.

Syntax

```
isAssigned ( ) Logical
```

Description

isAssigned returns True if a Query variable has been assigned a value; otherwise, it returns False. This method does not check the validity of the assigned query.

Example

{button ,AL(`OPAL_TYPE_QUERY;OPAL_METH_QUERY;',0,"Defaultoverview",)} Related Topics

isAssigned example

The following example uses **isAssigned** to determine if *qVar* had been assigned a value. Although the value is not a valid query, isAssigned returns True.

```
method pushButton(var eventInfo Event)
var
   qVar Query
endVar


qVar = Query

      This is not a query

      endQuery

msgInfo("Assigned?", qVar.isAssigned())    ; displays True

endMethod
```

isCreateAuxTables method

Reports whether the use of auxiliary tables is enabled.

Syntax

```
isCreateAuxTables ( ) Logical
```

Description

isCreateAuxTables reports whether the use of auxiliary tables is enabled. If **isCreateAuxTables** returns True, auxiliary tables are used to create a query's answer table.

Example

{button ,AL(`OPAL_TYPE_QUERY;OPAL_METH_QUERY;OPAL_METH_QUISCREATEAUXTABLES;',0,"Defaultoverview",)} Related Topics

isCreateAuxTables example

The following example contains a query that uses auxiliary tables:

```
method pushButton(var eventInfo Event)
var
myQBE Query
endvar


myQBE = Query


      Customer.db  |  Name        |
      Delete       |  Johnson..   |


EndQuery


if myQBE.isCreateAuxTables() = False then
   myQBE.createAuxTables(True)
else
endif
myQBE.executeQBE()
endMethod
```

isEmpty method

Determines whether the query is empty.

Syntax

```
isEmpty ( ) Logical
```

Description

isEmpty returns a Logical value indicating whether the query is empty. This method determines whether you have added anything to your query but does not determine if the query contains enough information to be run. For example, you can append an empty row to a query and use **isEmpty** to determine if the query is empty. **IsEmpty** returns False because you have appended pieces of the query. **isQueryValid** also returns False, because you did not complete the query.

 **Example**

{button ,AL(`OPAL_TYPE_QUERY;OPAL_METH_QUERY;OPAL_METH_QUISQUERYVALID;',0,"Defaultoverview",)} Related Topics

isEmpty example

The following example reports if the Query variable is empty, before and after a query by example (QBE) file is read into the Query variable. The Query variable is empty before it has been assigned a value. If the readFromFile method is successful, the Query variable contains data.

```
method pushButton(var eventInfo Event)
var
   qVar  Query
endVar


msgInfo( "Before readFromFile", "Query is " +
         iif(qVar.isEmpty(), "empty", "not empty"))
qVar.readFromFile("MyQuery.QBE")
msgInfo( "After readFromFile", "Query is " +
         iif(qVar.isEmpty(), "empty", "not empty"))
endMethod
```

isExecuteQBELocal method

Reports whether a query by example (QBE) was executed locally or on a server.

Syntax

```
isExecuteQBELocal ( ) Logical
```

Description

isExecuteQBELocal returns True if the query was executed locally; otherwise, it returns False. This method is especially useful when the server uses a different character set, sort order, or other feature that affects the query's result.

Example

{button ,AL(`OPAL_TYPE_QUERY;OPAL_METH_QUEXECUTEQBE;',0,"Defaultoverview",)} Related Topics

isExecuteQBELocal example

The following example calls **isExecuteQBELocal** to determine whether a query by example (QBE) was executed locally or on a server.

```
method pushButton (var eventInfo Event)
   var
      qbeVar        Query
      dlgTitleText,
      dlgBodyText   String
     endVar


    dlgTitleText = "Remote query"
    dlgBodyText  = "This query was not run on the server. \n" +
                   "Check the sort order"


    qbeVar = Query

             :WestData:orders.db |CustName|Qty        |
                                 |Check   |Check > 10 |


             endQuery


   if qbeVar.executeQBE() then
      if qbeVar.isExecuteQBELocal() then
         msgInfo(dlgTitleText, dlgBodyText)
      endIf
   else
         errorShow()
   endIf

endMethod
```

isQueryValid method

Compiles the current query and indicates whether it contains errors that prevent it from being run.

Syntax

```
isQueryValid ( ) Logical
```

Description

isQueryValid compiles the current query and indicates whether it contains errors that prevent it from being run. This is the same procedure that occurs when you interactively save a query to disk, execute a query, or request a query string. **isQueryValid** returns False if the query contains an error. To get information on the error, use **errorCode** (System type).

Example

{button ,AL(`OPAL_TYPE_QUERY;OPAL_METH_QUERY;OPAL_METH_QUGETCRITERIA;OPAL_METH_QUHASCRITERIA;OPAL_METH_QUREMOVECRITERIA;OPAL_METH_QUSETCRITERIA;',0,"Defaultoverview",)}

Related Topics

isQueryValid example

The following example creates a query and reports an error if **isQueryValid** returns False:

```
method pushButton(var eventInfo Event)
var
   qVar Query
   orderID SmallInt
endVar
orderID = qVar.appendTable( "Orders.db" )
qVar.setCriteria( orderID, "Sale Date", "> 1/1/95" )
if not qVar.isQueryValid() then
      errorShow()   ; note that no fields are checked
endif
endMethod
```

query keyword

Marks the beginning of a query statement.

Syntax

```
query
   tableName|fieldName|[ fieldName|] *
            |criteria |[ criteria |] *
 [ tableName|fieldName|[ fieldName|] *
            |criteria |[ criteria |] * ] *
endQuery
```

Description

query marks the beginning of a query by example (QBE) statement, which assigns a query to a Query variable. A QBE statement extracts data from one or more tables according to the fields specified in *fieldName* and the selection criteria (*criteria* can be any valid QBE expression). Because this type of query is not a string, it can contain tilde variables. For more information, see **readFromString**.)

A query statement contains a Query variable, the = sign, and the keyword query followed by a blank line. The body of the query is followed by another blank line, and the query ends with the keyword endQuery.

Note

- You don't have to list all the fields in the table. The following example lists only those fields that affect the query:

```
var myQBE Query endvar
myQBE = Query

             Customer|Customer No|Name  |
                     |Check      |A..  |

          endQuery
```

This query statement retrieves customer numbers whose names start with A from the Customer table. Only two of the Customer table's fields are specified.

You can align the vertical field separators to make the code more readable; however, ObjectPAL also recognizes the following code:

```
var myQBE Query endvar
myQBE = Query

Customer|Customer No           |Name  |
|Check| A..  |

    endQuery
```

If you construct a query statement that includes two or more tables, you must separate each table with a blank line. The following code example separates the Customer and Orders tables with a blank line:

```
var myQBE Query endvar
myQBE = Query

            Customer|Customer No|Name |Phone |
                    |_x         |Check|Check |

            Orders  |Customer No|Balance Due|
                    |_x         |Check 0    |

         endQuery
```

You can use absolute paths or aliases to specify where to find tables in the query definition. Corel Paradox also searches for unqualified table names (i.e, table names without paths or aliases) in a specified database. If a database is not specified, Corel Paradox searches the default database (the working directory).

Example

{button ,AL(`OPAL_TYPE_QUERY;OPAL_METH_QUEXECUTEQBE;OPAL_METH_QUREADFROMFILE;OPAL_METH_QUREADFROMSTRING;OPAL_METH_QUWRITEQBE;',0,"Defaultoverview",)} Related Topics

query example

The following example uses the **pushButton** method for the *getReceivables* button to construct a query statement, assign it to a Query variable and run it with **executeQBE**. In this example, the query statement is an insert query; it retrieves records from CUSTOMER.DB and ORDERS.DB and inserts them into the existing *MyCust* table. The selection criteria uses a tilde variable called *myState* that includes Oregon customers in the results. Since OR is the abbreviation for Oregon, the *myState* variable must evaluate to a quoted string to distinguish between the selection criteria and the **OR** query expression.

```
method pushButton(var eventInfo Event)
var
   qVar    Query
   myState String
   tv      TableView
endVar


; add samp alias for the sample directory
addAlias("samp", "Standard", "c:\\Corel\\Suite8\\Paradox\\samples")


; OR is the abbreviation for Oregon, but because it's
; also an ObjectPAL keyword, it must be enclosed in quotes.
myState = "\"OR\""


; now use myState as a tilde variable in this query statement
qVar = Query


      :samp:Customer.db|Customer No|Name |State/Prov|Phone  |
                       |_cust      |_name|  ~myState|_phone |


       :samp:Orders.db |Customer No|Balance Due|
                       |_cust      |0, _balDue |


             myCust.db |Customer No|Name |Balance Due|Phone  |
               insert  |_cust      |_name|_balDue    |_phone |


        EndQuery


qVar.executeQBE("myCust.db")   ; put results into myCust.db
tv.open("myCust.db")            ; view the table


endMethod
```

readFromFile method

Assigns the contents of a query by example (QBE) file to a Query variable.

Syntax

```
readFromFile ( const qbeFileName String ) Logical
```

Description

readFromFile opens *qbeFileName* and assigns the contents to a Query variable. There are several ways to create a query file; for example, in ObjectPAL using **writeQBE**, or interactively using the Query Editor. Use **executeQBE** to execute the query.

If the value of *qbeFileName* does not include a path or alias, **readFromFile** searches for the file in the directory associated with a specified database. If a database is not specified, **readFromFile** searches the default database. If the value of *qbeFileName* does not include an extension, this method assumes an extension of .QBE. To specify a filename that does not have an extension, add a period to the end of the name. For example, the following table lists the resulting filenames for various values of *qbeFileName*.

Value of *qbeFileName*	QBE filename
newcust	newcust.qbe
newcust.	newcust
newcust.q	newcust.q

readFromFile returns True if it succeeds; otherwise, it returns False.

Example

{button ,AL(`OPAL_TYPE_QUERY;OPAL_METH_QUREADFROMSTRING;',0,"Defaultoverview",)} Related Topics

readFromFile example

The following example reads and executes the query.

```
method pushButton(var eventInfo Event)
var
   qVar   Query
endVar


 ; this writes results into :PRIV:ANSWER.DB
qVar.readFromFile("GetCust.qbe")
qVar.executeQBE()


endMethod
```

readFromString method

Assigns a query string to a Query variable.

Syntax

```
readFromString ( const QBEString String ) Logical
```

Description

readFromString assigns the query string specified in *QBEString* to a Query variable. Use **executeQBE** to execute the query.

Use **readFromString** to build a QBE string from smaller strings. a QBE string can be a combination of quoted strings and string variables.

You can use absolute paths or aliases to specify where to find tables in the query definition. Corel Paradox also searches for unqualified table names (i.e., table names without paths or aliases) in a specified database. If a database is not specified, Corel Paradox searches the default database (the working directory). Double backslashes are required when specifying a path.

Because a QBE string is a quoted string, it cannot contain tilde variables; however you can use string variables to achieve the same effect. To include tilde variables in a query, use a **query** statement.

Example

{button ,AL(`OPAL_TYPE_QUERY;OPAL_METH_QUEXECUTEQBE;OPAL_METH_QUREADFROMFILE;',0,"Defaultoverview",)} **Related Topics**

readFromString example

The following example uses the **pushButton** method for *btnFindName* to define a query as a string value and then uses **readFromString** to assign the string to a Query variable:

```
method pushButton(var eventInfo Event)
var
   db Database
   qs String
   tv TableView
   tc TCursor
   qVar   Query
endVar


; Add the sampData alias then open the database.
addAlias("sampData", "Standard", "c:\\Corel\\Suite8\\Paradox\\samples")
db.open("sampData")


; Open a TCursor for the Stock table.
tc.open("Stock.db", db)


; If locate finds Krypton Flashlight in the Description field.
if tc.locate("Description", "Krypton Flashlight") then


   ; Now use the Stock No field value in Stock.db in a query string.
   qs = "Query\n\n" +
       ":sampData:Lineitem|Order No|Stock No |\n" +
                          "| _ordNo |" + tc."Stock No" + "|\n\n" +
         ":sampData:Orders|Order No|Customer No |\n" +
                          "| _ordNo|_cust |\n\n" +
       ":sampData:Customer|Customer No|Name|Phone |\n" +
                          "| _cust|Check|Check |\n\n" +
        "EndQuery"


   ; Note that the vertical lines (|) don't have to be aligned.


   qVar.readFromString(qs)


   if qVar.executeQBE() then
      tv.open(":priv:answer.db")       ; Display the answer table.
   else
      msgStop("Error", "Query failed") ; Otherwise, query failed.
   endIf


else
   msgStop("Error", "Can't find Krypton Flashlight")
endIf


endMethod
```

removeCriteria method

Removes the query expression from a specified field.

Syntax

```
removeCriteria ( const tableID SmallInt, const fieldID SmallInt ) Logical
removeCriteria ( const tableID SmallInt, const fieldName String ) Logical
removeCriteria ( const tableID SmallInt, const rowID SmallInt, const fieldID SmallInt ) Logical
removeCriteria ( const tableID SmallInt, const rowID SmallInt, const fieldName String ) Logical
removeCriteria ( const tableName String, const fieldID SmallInt ) Logical
removeCriteria ( const tableName String, const fieldName String ) Logical
removeCriteria ( const tableName String, const rowID SmallInt, const fieldID SmallInt ) Logical
removeCriteria ( const tableName String, const rowID SmallInt, const fieldName String ) Logical
```

Description

removeCriteria removes the query expression in a specified field, but does **not** remove check marks. Use **setCheck** and **clearCheck** to manipulate query check marks.

The table is specified numerically by *tableID* or by *tableName*. The field is specified by the *fieldID* or by *fieldName*. The row is specified by the row identifier *rowID*. If no row is specified, this method defaults to the first row.

Example

{button ,AL(`OPAL_TYPE_QUERY;OPAL_METH_QUERY;OPAL_METH_QUGETCRITERIA;OPAL_METH_QUHASCRITERIA;OPAL_METH_QUISQUERYVALID;OPAL_METH_QUSETCRITERIA;',0,"Defaultoverview",)} Related Topics

removeCriteria example

The following example removes the criteria from the Name field in the CUSTOMER.DB table in the query:

```
method pushButton(var eventInfo Event)
var
   qVar Query
endVar


   qVar.readFromFile( "Myquery.qbe" )
   qVar.removeCriteria( "Customer.db" , "Name" )
   qVar.executeQBE()      ; execute the saved query minus the
                          ; customer name criteria.
endMethod
```

removeRow method

Deletes a row and its contents from the query workspace.

Syntax

```
removeRow ( const rowID SmallInt, const tableID SmallInt ) Logical
removeRow ( const rowID String, const tableName SmallInt ) Logical
```

Description

removeRow deletes a row and its contents from a query.

The table is specified numerically by *tableID* or by *tableName*. The row is specified by the row identifier *rowID*.

Example

{button ,AL(`OPAL_TYPE_QUERY;OPAL_METH_QUERY;OPAL_METH_QUAPPENDTABLE;OPAL_METH_QUINSERTROW;OPAL_METH_QUINSERTTABLE;OPAL_METH_QUREMOVETABLE;OPAL_METH_QUSETROWOP;',0,"Defaultoverview",)} Related Topics

removeRow example

The following example removes the second row from the ORDER.DB table in MYQUERY.QBE:

```
method pushButton(var eventInfo Event)
var
   qVar Query
   rowID SmallInt
endVar
   qVar.readFromFile( "MyQuery.qbe" )
   rowID = qVar.getRowID( "ORDERS.DB", 2 )  ; get the 2nd row
   qVar.removeRow( "ORDERS.DB", rowID )
   qVar.executeQBE()
endMethod
```

removeTable method

Deletes a table from the query.

Syntax

```
removeTable ( const tableID SmallInt ) Logical
removeTable ( const tableName String ) Logical
```

Description

removeTable deletes a table from the query. The table is specified numerically by *tableID* or by *tableName*.

Example

{button ,AL(`OPAL_TYPE_QUERY;OPAL_METH_QUERY;OPAL_METH_QUAPPENDTABLE;OPAL_METH_QUINSERTROW;OPAL_METH_QUINSERTTABLE;OPAL_METH_QUREMOVEROW;OPAL_METH_QUSETROWOP;',0,"Defaultoverview",)} **Related Topics**

removeTable example

The following example removes the table ORDERS.DB from the query image MYQUERY.QBE:

```
method pushButton(var eventInfo Event)
var
   qVar Query
endVar


   qVar.readFromFile( "MyQuery.qbe" )
   qVar.removeTable( "Orders.db" )          ; remove Orders.db from
                                            ; the workspace
   qVar.removeCriteria("Customer.db", "Customer No" )  ; clear the
                                            ; example element link.
   qVar.executeQBE()
endMethod
```

setAnswerFieldOrder method

Sets the field order of the answer table that is generated by a query.

Syntax

```
setAnswerFieldOrder ( var fieldOrder Array[] String ) Logical
```

Description

setAnswerFieldOrder sets the field order of the answer table generated by a query. The parameter *fieldOrder* is an array of field names that can be used the answer table structure. Use **getAnswerName t**o retrieve an array of these field names and then modify the order.

If you retrieve an array of field names and change the answer field list (e.g., by unchecking a field), the array is instantly out of date. You must remove the modified field from your array before using the array for field ordering. A specified field order must contain the same number of elements as fields in the answer table.

Example

{button ,AL(`OPAL_TYPE_QUERY;OPAL_METH_QUGETANSWERFIELDORDER;OPAL_METH_QUGETANSWERNAME;OPAL_METH_QUGETANSWERSORTORDER;OPAL_METH_QUERY;OPAL_METH_QUSETANSWERNAME;OPAL_METH_QUSETANSWERSORTORDER;',0,"Defaultoverview",)} Related Topics

setAnswerFieldOrder example

The following example retrieves the field names from MYQUERY.QBE, changes their order, and uses **setAnswerFieldOrder** to put the new order in place.

```
method pushButton(var eventInfo Event)
var
   qVar  Query
   arFields  Array[] String
endVar
   qVar.readFromFile( "myquery.qbe" )
   qVar.getAnswerFieldOrder( arFields )
      if arFields.size() > 0 then     ; swap the first and third fields
                                      ; in the answer table.
         arFields.exchange(1,3)
         qVar.setAnswerFieldOrder( arFields )
         qVar.executeQBE()
      endif
endMethod
```

setAnswerName method

Sets the name of the answer table that is generated by a query.

Syntax

```
setAnswerName ( const tableName String ) Logical
```

Description

setAnswerName specifies *tableName* as the name of the answer table that is created by the query.

Example

{button ,AL(`OPAL_TYPE_QUERY;OPAL_METH_QUGETANSWERFIELDORDER;OPAL_METH_QUGETANSWERNAME;OPAL_METH_QUGETANSWERSORTORDER;OPAL_METH_QUERY;OPAL_METH_QUSETANSWERFIELDORDER;OPAL_METH_QUSETANSWERSORTORDER;',0,"Defaultoverview",)} Related Topics

setAnswerName example

The following example allows the user to change the answer table name for MYQUERY.QBE:

```
method pushButton(var eventInfo Event)
var
   qVar  Query
   AnsTblName String
endVar
if msgQuestion("Query",
               "Would you like to change the "
             + "answer table name?") = "Yes" then
   qVar.readFromFile("MYQUERY.QBE")
   AnsTblName = qVar.getAnswerName()
   AnsTblName.view("Make changes below")
   qVar.setAnswerName(AnsTblName)
   qVar.writeQBE("MYQUERY.QBE")
endif
endMethod
```

setAnswerSortOrder method

Specifies the sort order for fields in the answer table.

Syntax

```
setAnswerSortOrder ( var sortFields Array[] String ) Logical
```

Description

setAnswerSortOrder specifies the sort order for fields in the answer table. The array *sortFields* contains an ordered list of field names. Use **getAnswerSortOrder** to retrieve this array and then change the field name positions to create a new sort order.

If you retrieve an array of field names and change the answer field list (e.g., by unchecking a field), the array is instantly out of date. You must remove the modified field from your array before using the array for field ordering.

 **Example**

{button ,AL(`OPAL_TYPE_QUERY;OPAL_METH_QUGETANSWERFIELDORDER;OPAL_METH_QUGETANSWERNAME;OPAL_METH_QUGETANSWERSORTORDER;OPAL_METH_QUERY;OPAL_METH_QUSETANSWERFIELDORDER;OPAL_METH_QUSETANSWERNAME;',0,"Defaultoverview",)} Related Topics

setAnswerSortOrder example

The following example retrieves the field list from MYQUERY.QBE, reorders the fields, and uses **setAnswerSortOrder** to put the new order in place:

```
method pushButton(var eventInfo Event)
var
   qVar  Query
   arFields  Array[] String
endVar
   qVar.readFromFile( "myquery.qbe" )
   qVar.getAnswerSortOrder( arFields )
      if arFields.size() > 0 then     ; swap the first and third fields
                                      ; in the sort order.
         arFields.exchange(1,3)
         qVar.setAnswerSortOrder( arFields )
         qVar.executeQBE()
      endif
endMethod
```

setCriteria method

Specifies the criteria for a table's field.

Syntax

setCriteria (const *tableID* SmallInt, const *fieldID* SmallInt, const *newCriteria* String) Logical

setCriteria (const *tableID* SmallInt, const *fieldName* String, const *newCriteria* String) Logical

setCriteria (const *tableID* SmallInt, const *rowID* SmallInt, const *fieldID* SmallInt, const *newCriteria* String) Logical

setCriteria (const *tableID* SmallInt, const *rowID* SmallInt, const *fieldName* String, const *newCriteria* String) Logical

setCriteria (const *tableName* String, const *fieldID* SmallInt, const *newCriteria* String) Logical

setCriteria (const *tableName* String, const *fieldName* String, const *newCriteria* String) Logical

setCriteria (const *tableName* String, const *rowID* SmallInt, const *fieldID* SmallInt, const *newCriteria* String) Logical

setCriteria (const *tableName* String, const *rowID* SmallInt, const *fieldName* String, const *newCriteria* String) Logical

Description

setCriteria specifies a query expression string to be used as the criteria for a specific table's field. The table is specified numerically by *tableID* or by *tableName*. The field is specified by the *fieldID* or by *fieldName*. The row is specified by the row identifier *rowID*. If no row is specified, this method defaults to the first row. The criteria is specified by *newCriteria*.

setCriteria does **not** support check marks.

Example

{button ,AL(`OPAL_TYPE_QUERY;OPAL_METH_QUERY;OPAL_METH_QUGETCRITERIA;OPAL_METH_QUHASCRITERIA;OPAL_METH_QUISQUERYVALID;OPAL_METH_QUREMOVECRITERIA;',0,"Defaultoverview",)}

Related Topics

setCriteria example

The following example sets the criteria for the appended row (State/Prov) to either CA or HI.

```
method pushButton(var eventInfo Event)
var
   qVar Query
   rowID SmallInt
endVar


   qVar.appendTable( "CUSTOMER.DB" )
   rowID = qVar.appendRow( "CUSTOMER.DB" )
   qVar.setCriteria( "CUSTOMER.DB", rowID, "State/Prov", "CA or HI" )
   qvar.checkRow("Customer.db", rowID, CheckCheck)
   qvar.writeQBE("MyQBE")
endMethod
```

setLanguageDriver method

Specifies the name of the default language driver for your system.

Syntax

```
setLanguageDriver ( const languageDriver String ) Logical
```

Description

setLanguageDriver specifies the default language driver to the driver specified by *languageDriver*. The language driver is a part of the table's definition. The language drivers for Corel Paradox tables are listed in of the description of the Table type's **create** method.

If you execute a query on a table that uses a different language driver you can use System type's **getLanguageDriver** to identify the language driver of the table. Set the language driver for the query using **setLanguageDriver**, to create the query's answer table with the same driver.

Example

{button ,AL(`OPAL_TYPE_QUERY;OPAL_METH_SYGETLANGUAGEDRIVER;OPAL_METH_SYSYSINFO;OPAL_METH_TBCREA;OPAL_METH_TCGETLANGUAGEDRIVER;',0,"Defaultoverview",)} Related Topics

setLanguageDriver example

The following example sets the language driver to Czech:

```
method pushButton(var eventInfo Event)
var
myQBE Query
endvar


myQBE = Query


   Customer|Customer No | Name  |
           |Check       | A..  |


endQuery
myQBE.setLanguageDriver ("ANCZECH")
myQBE.executeQBE()
endMethod
```

setQueryRestartOptions method

Specifies the function of the underlying tables while running a query.

Syntax

```
setQueryRestartOptions ( const qryRestartType SmallInt ) Logical
```

Description

setQueryRestartOptions tells Corel Paradox what to do if data changes while you're running a query in a multi-user environment. The argument *qryRestartType* represents one of the following ObjectPAL QueryRestartOptions constants:

QueryDefault	Use the options specified interactively by using the Query Restart Options dialog box.
QueryLock	Lock other users out of the tables needed to run the query. If Corel Paradox cannot lock a table, it does not run the query.
QueryNoLock	Run the query even if the data changes while it's running.
QueryRestart	Restart the query. Specify QueryRestart to get a snapshot of the data as it existed at a particular instant.

Examples

{button ,AL(`OPAL_TYPE_SQL;OPAL_TYPE_QUERY;OPAL_METH_QUGETQRESTART;',0,"Defaultoverview",)}
Related Topics

setQueryRestart examples

Example1 Using **executeSQL**
Example2 Using **executeQBE**

setQueryRestartOptions example 1

The following example calls **getQueryRestartOptions** to retrieve the user's query restart options. The code uses **executeSQL**. If the setting is not QueryRestart, this code calls **setQueryRestartOptions** to set it before executing the query:

```
method pushButton(var eventInfo Event)
   var
      qVar   SQL
      MyDB  database
   endVar


  MyDB.open("work")


  if qVar.getQueryRestartOptions() <> QueryRestart then
     setQueryRestartOptions(QueryRestart)
  endIf


   if qVar.readFromFile("newcust.sql") then
      qVar.executeSQL(MyDB)
   else
      errorShow()
   endIf
endMethod
```

setQueryRestartOptions example 2

The following example calls **getQueryRestartOptions** to retrieve the current query restart options. The code uses **executeQBE**. If the setting is not QueryRestart, this code calls **setQueryRestartOptions** to set it and executes the query:

```
method pushButton(var eventInfo Event)
   var
      qVar   Query
   endVar


   if getQueryRestartOptions() <> QueryRestart then
      setQueryRestartOptions(QueryRestart)
   endIf


   if qVar.readFromFile("newcust.qbe") then
      qVar.executeQBE()
   else
      errorShow()
   endIf


endMethod
```

setRowOp method

Sets the row operator for a specific row.

Syntax

```
setRowOp ( const tableID SmallInt, const rowID SmallInt, const rowOperator SmallInt) Logical
setRowOp ( const tableID SmallInt, const rowOperator SmallInt) Logical
setRowOp ( const tableName String, const rowID SmallInt, const rowOperator SmallInt) Logical
setRowOp ( const tableName String, const rowOperator SmallInt) Logical
```

Description

setRowOp sets one of the four row operators in a specified row. The table is specified numerically by *tableID* or by *tableName*. The row is specified by the row identifier *rowID*. If no row is specified, this method defaults to the first row.

The ***rowOperator*** is one of the following values:

qbeRowDelete	Delete operator
qbeRowInsert	Insert operator
qbeRowNone	No operator
qbeRowSet	Set operator

Example

{button ,AL(`OPAL_TYPE_QUERY;OPAL_METH_QUERY;OPAL_METH_QUAPPENDTABLE;OPAL_METH_QUINSERTROW;OPAL_METH_QUINSERTTABLE;OPAL_METH_QUREMOVEROW;OPAL_METH_QUREMOVETABLE;',0,"Defaultoverview",)} Related Topics

setRowOp example

The following example deletes records with blank Customer No. fields.

```
method pushButton(var eventInfo Event)
var
   qVar  Query
endVar


   qVar.appendTable( "Customer.db" )
   qVar.setRowOp( "Customer.db" , qbeRowDelete)
   qVar.setCriteria( "Customer.db" , "Customer No" , "blank" )
                                  ; delete blank Customer No records.
   qVar.executeQBE()

endMethod
```

wantInMemoryTCursor method

Specifies how to create a TCursor resulting from a query.

Syntax

```
wantInMemoryTCursor ( [ const yesNo Logical ] )
```

Description

wantInMemoryTCursor specifies how to create a TCursor resulting from a query. When you call **wantInMemoryTCursor** with *yesNo* set to as Yes or omitted, Corel Paradox creates a dead TCursor in system memory, with no connection to underlying tables. When *yesNo* is No, Corel Paradox creates a TCursor in a live query view. By default, when you execute a query to a TCursor, that TCursor will point to a live query view. changes made to the TCursor will affect the underlying tables. Set **wantInMemoryTCursor** to Yes when you *don't* want a live query view.

An in-memory TCursor can be useful for performing quick analyses. Suppose you want to study the effects of giving each employee a 15 percent raise. Query the employee data to increase everyone's salary by 15 percent and execute the query to an in-memory TCursor. Now you can work with the queried data there, without affecting the actual employee data.

Example

{button ,AL(`OPAL_TYPE_QUERY;OPAL_METH_QUEXECUTEQBE;OPAL_METH_SQLWANTINMEMORYTCURSOR;OPAL_METH_TCINSTANTIATEVIEW;OPAL_METH_TCISINMEMORYTCURSOR;OPAL_METH_TCISVIEW;',0,"Defaultoverview",)} Related Topics

wantInMemoryTCursor example

The following example uses an in-memory TCursor to study the effects of giving every employee a 15 percent raise. The code reads and executes a predefined query and then uses the results in a calculation:

```
method pushButton(var eventInfo Event)
   var
      qVar            Query
      tcRaise15       TCursor
      nuTotalPayroll  Number
   endVar


   qVar.wantInMemoryTCursor(Yes)
   qVar.readFromFile("raise15.qbe")
   qVar.executeQBE(tcRaise15)


   nuTotalPayroll = tcRaise15.cSum("Salary")
   nuTotalPayroll.view("Payroll after 15%   raise:")


endMethod
```

writeQBE method/procedure

Writes a query statement to a specified file.

Syntax

```
1. (Method) writeQBE ( const fileName String ) Logical
2. (Procedure) writeQBE ( const str String , const fileName String ) Logical
```

Description

writeQBE writes a predefined query to the file specified in *fileName*. If *fileName* exists, this method overwrites it without asking for confirmation. If *fileName* does not specify a path, Corel Paradox writes to the working directory. **writeQBE** returns True if the write succeeds; otherwise it returns False.

Syntax 1 calls **writeQBE** as a method. It writes the query represented by an assigned Query variable to the file specified in *fileName*.

Syntax 2 calls **writeQBE** as a procedure. It writes the query string represented by *str* to the file specified in *fileName*.

Example

{button ,AL(`OPAL_TYPE_QUERY;OPAL_METH_QUEXECUTEQBE;OPAL_METH_QUREADFROMFILE;OPAL_METH_QUREADFROMSTRING;',0,"Defaultoverview",)} Related Topics

writeQBE example

The following example assumes that a form contains a button named *getDest*. When the form opens, this code determines whether the GETDEST.QBE file exists in the current directory. If the file does not exist, the built-in **open** method for *pageOne* uses **writeQBE** to write a query string to GETDEST.QBE. The built-in **pushButton** for *getDest* runs the query and then opens the table. This example assumes that the :MAST: alias has already been defined.

The following code is attached to the **open** method for *pageOne*:

```
method pushButton(var eventInfo Event)
Var
   qVar Query
endVar


; if the GetDest.qbe query file doesn't exist
if not isFile("GetDest.qbe") then


   ; construct a query
   qVar = Query


            :mastApp:Dest|Destination Name|Avg Temp (F)|
                         |Check           |Check 70    |


          EndQuery


   ; write the query statement to the GetNames.qbe file
   qVar.writeQBE("GetDest.qbe")


endIf
endMethod
```

The following code is attached the built-in **pushButton** method for the *getDest* button. This code does not check whether GETDEST.QBE exists because the **open** method for the page ensures that the file is available.

```
method pushButton(var eventInfo Event)
var
   qVar   Query
   tv    TableView
endVar


qVar.readFromFile("GetDest.qbe")
qVar.executeQBE("MyDest")
tv.open("MyDest")


endMethod
```

You can also use **writeQBE** method with ObjectPAL to create and save a query that your user can run interactively, using the Query Editor.

Record type

ObjectPAL's Record type is a programmatic, user-defined collection of information that resembles a **record** in Pascal or a **struct** in C. Records that are defined in ObjectPAL code are separate and distinct from records associated with a table.

The following code declares a Record data type:

```
TYPE
recordName = RECORD
                    fieldName fieldType
                  [ fieldName fieldType ] *
             ENDRECORD
ENDTYPE
```

fieldName identifies fields or columns in the record, and *fieldType* specifies one of the data types. Records are declared in a design object's Type window.

After you declare a Record data type, you can use the = and <> comparison operators to compare records. You can also use the (=) assignment operator to copy a record's contents to another record.

Several predefined record structures have been created for specific situations. For more information, see the following topics:

- FormOpenInfo (see **Open** (Form type)
- ReportOpenInfo (see **Open** (Report type)
- FileBrowserInfo (see **FileBrowser** (System type)
- PrinterOptionInfo, PrinterGetOptions (Sytem type), and PrinterSetOptions (System type).

When declaring a record variable of these structures, use the predefined structure name instead of declaring the variable as a Record type. For example, if you want to declare a variable called *MySettings* which has the predefined structure of FormOpenInfo, do the following:

```
var
     MySettings       FormOpenInfo        ; note that you do not declare MySettings as type
Record
endvar
```

ObjectPAL automatically creates the *MySettings* variable with the predefined structure for a FormOpenInfo record. Any Record methods can be used with *MySettings.*

The Record type includes several derived methods from the AnyType type.

Methods for the Record type

AnyType	**Record**
blank	**view**
dataType	
isAssigned	
isBlank	
isFixedType	

Print related ObjectPAL methods and examples

view method

Displays the value of a variable in a dialog box.

Syntax

```
view ( [ const title String ] )
```

Description

view displays the value or values assigned to a Record variable in a modal dialog box. ObjectPAL execution suspends until the user closes this dialog box. You can specify the dialog box's title in *title*, or you can omit *title* to display the variable's data type.

Note

- Values in a Record can't be changed when displayed in a **view** dialog box. For more information, see AnyType.

Example

{button ,AL(`OPAL_TYPE_RECORD;OPAL_TYPE_ARRAY;OPAL_TYPE_DYNARRAY;',0,"Defaultoverview",)}

Related Topics

view example

The following example uses the **pushButton** method for *getAndViewRec* to declare a variable called *myRec* of the MyRecord type. This method opens a TCursor to the *Customer* table, fills *myRec* with the *Customer No* and *Name* field values, and uses **view** to display the record in a dialog box. The operation is then repeated for the second record in *Customer*.

The following code is attached to the Type window for *getAndViewRec* to create a user-defined type named MyRecord:

```
; getAndViewRec::Type
Type
  MyRecord = RECORD         ; define a Record structure
               ID    String
               Name  String
             ENDRECORD
endType
```

The following code is attached to the **pushButton** method for a button named *getAndViewRec*:

```
; getAndViewRec::pushButton
method pushButton(var eventInfo Event)
var
  recOne, recTwo MyRecord
  tc             TCursor
endVar


if tc.open("Customer.db") then
  recOne.ID = tc."Customer No"     ; put some values into the record
  recOne.Name = tc."Name"
  recOne.view("First record")      ; display the record in a dialog box


  tc.nextRecord()                  ; move to the next record


  recTwo.ID = tc."Customer No"     ; get new values
  recTwo.Name = tc."Name"
  recTwo.view("Second record")     ; display second record


  msgInfo("recOne = recTwo?", recOne = recTwo)  ; displays False


  recOne = recTwo                  ; now both records have the same values
  msgInfo("recOne = recTwo?", recOne = recTwo)  ; displays True


else
  msgStop("Stop", "Couldn't open the Customer table.")
endIf
endMethod
```

Report type

A Report variable provides a handle to a report. You use Report variables in code to manipulate the report onscreen. Report methods control the window's size, position, and appearance, and allow you to view and print the report.

Use **load** to load a report file in the Report Design window; use **open** to open the report in the Report window, and use **print** to open a report and print it. You cannot attach methods to objects in a report but you can attach code to calculated fields.

The foloowing table displays the methods for the Report type, including several derived methods from the Form type.

Methods for the Report type

Form	**Report**
bringToTop	**attach**
create	**close**
deliver	**currentPage**
dmAddTable	**design**
dmBuildQueryString	**enumUIObjectNames**
dmEnumLinkFields	**enumUIObjectProperties**
dmGetProperty	**load**
dmHasTable	**moveToPage**
dmLinkToFields	**open**
dmLinkToIndex	**print**
dmRemoveTable	**run**
dmSetProperty	**setMenu**
dmUnlink	
enumDataModel	
enumSource	
enumTableLinks	
getFileName	
getPosition	
getProtoProperty	
getStyleSheet	
getTitle	
hide	
isDesign	
isMaximized	
isMinimized	
isVisible	
maximize	
menuAction	
minimize	
moveTo	
save	
saveStyleSheet	
selectCurrentTool	
setIcon	
setPosition	
setProtoProperty	
setSelectedObjects	
setStyleSheet	

setTitle
show
wait
windowClientHandle
windowHandle

Print related ObjectPAL methods and examples

attach method

Associates a Report variable with an open report.

Syntax

```
attach ( const reportTitle String ) Logical
```

Description

attach associates a Report variable with the open report. *reportTitle* specifies the title of an open report.

Note

- The argument *reportTitle* refers to the text displayed in the Title Bar of the Report window (not to the filename). You can use **getTitle** to return this text, or you can use **setTitle** to specify a new title.

Example

{button ,AL(`OPAL_TYPE_REPORT;OPAL_METH_REOPEN;OPAL_METH_FOGETT;OPAL_METH_FOSTIT;',0,"Defaultoverview",)} **Related Topics**

attach example

In the following example, assume the form's **open** method opened the STOCK.RSL report and retitled the window as Stock Report. The **pushButton** method for *printStock* attaches to the open report and prints its content.

```
; printStock::pushButton
method pushButton(var eventInfo Event)
var
  stockRep  Report
endVar
; the Stock report was opened and retitled by the form's open method
stockRep.attach("Stock Report")  ; attach by report's title
stockRep.print()                 ; print the report
endMethod
```

This code is attached to the form's **open** method:

```
; thisForm::open
method open(var eventInfo Event)
var
  stockRep  Report
endVar
if eventInfo.isPreFilter()
  then
    ;code here executes for each object in form
  else
    ;code here executes just for form itself
    stockRep.open("stock.rsl")
    stockRep.setTitle("Stock Report")
    bringToTop()          ; bring this form back to the top
endIf
endMethod
```

close method

Closes a Report window.

Syntax

```
close ( )
```

Description

close closes a Report window. This method is the equivalent of choosing Close from the Control menu.

Example

{button ,AL(`OPAL_TYPE_REPORT;OPAL_METH_REOPEN;',0,"Defaultoverview",)} **Related Topics**

close example

The following example assumes that the form's **open** method opened the STOCK.RSL report and retitled the window as Stock Report. The **close** method for the form attaches to the open report and closes it when the form closes.

```
; thisForm::close
method close(var eventInfo Event)
var
  stockRep  Report
endVar
if eventInfo.isPreFilter()
  then
    ; code here executes for each object in form
  else
    ; code here executes just for form itself
    ; the Stock report was opened and retitled by
    ; the form's open method
    stockRep.attach("Stock Report")
    stockRep.close()
endIf
endMethod
```

currentPage method

Returns the report's current page number.

Syntax

currentPage ( ) SmallInt

Description

currentPage returns the report's current page number.

Example

{button ,AL(`OPAL_TYPE_REPORT;OPAL_METH_REMOTP;',0,"Defaultoverview",)} **Related Topics**

currentPage example

In the following example, the **pushButton** method for *plusTwoPages* attaches to an open report. If this fails, the code opens the report. When the *ordersRep* variable points to an open report, this code moves the report forward two pages.

```
; plusTwoPages::pushButton
method pushButton(var eventInfo Event)
var
  ordersRep Report
endVar
; report might be open already, so attempt an attach first
if NOT ordersRep.attach("Report : ORDERS.RSL") then
  if NOT ordersRep.open("Orders.rsl") then
    msgStop("FYI", "Could not open or attach to report.")
    return
  endIf
endIf
; move to two pages past the current page
ordersRep.moveToPage(ordersRep.currentPage() + 2)
bringToTop()   ; make this form the top layer again
endMethod
```

design method

Switches a report from a Report window to a Report Design window.

Syntax

```
design ( ) Logical
```

Description

design switches a report from the Report window to the Report Design window. This method works only with saved reports (.RSL) and not with delivered reports (.RDL).

Use **run** to switch from a Report Design window to a Report window. Use **load** to open a report in a Report Design window.

Note

- You might need to follow a call to **open**, **load**, **design**, or **run** with a **sleep**. For more information, see the **sleep** method in the System type.

Example

{button ,AL(`OPAL_TYPE_REPORT;OPAL_METH_RELOAD;OPAL_METH_REOPEN;OPAL_METH_RERUN;',0,"Defaultoverview",)} Related Topics

design example

In the following example, assume that the form's **open** method opened the STOCK.RSL report and retitled the window as Stock Report. The **pushButton** method for *stockDesign* attaches to the open report and switches the report to the Report Design window.

```
; stockDesign::pushButton
method pushButton(var eventInfo Event)
var
  stockRep  Report
endVar
; the form's open method opened and retitled the Stock report
stockRep.attach("Stock Report")
stockRep.design()            ; switch to Design mode
endMethod
```

enumUIObjectNames method

Creates a table listing the UIObjects in a report.

Syntax

```
enumUIObjectNames ( const tableName String ) Logical
```

Description

enumUIObjectNames creates a Corel Paradox table listing the name and type of objects contained in a specified report. Use the argument *tableName* to specify a name for the table. If *tableName* already exists, this method overwrites it without asking for confirmation. If *tableName* is already open, this method fails. You can include an alias or path in *tableName*; if no alias or path is specified, Corel Paradox creates *tableName* in the working directory.

The following table displays the structure of *tableName*:

Field	Type	Size
ObjectName	A	128
ObjectClass	A	32

Example

{button ,AL(`OPAL_TYPE_REPORT;OPAL_METH_REENPRO;',0,"Defaultoverview",)} Related Topics

enumUIObjectNames example

In the following example, the **pushButton** method for *describeReport* uses **enumUIObjectNames** and **enumUIObjectProperties** to document a report:

```
; describeReport::pushButton
method pushButton(var eventInfo Event)
var
  ordersRep Report
  tempTable TableView
endVar
ordersRep.load("Orders.rsl")                      ; load report in Report Design window
ordersRep.enumUIObjectNames("ordnames.db")        ; write names to table
ordersRep.enumUIObjectProperties("ordprops.db")   ; write properties to table
ordersRep.close()
tempTable.open("ordnames")                        ; observe your handiwork
tempTable.wait()
tempTable.open("ordprops")
tempTable.wait()
tempTable.close()
endMethod
```

enumUIObjectProperties method

Lists the properties of each UIObject in a report.

Syntax

```
enumUIObjectProperties ( const tableName String ) Logical
```

Description

enumUIObjectProperties creates a Corel Paradox table listing the name, property name, and property value of each object in a report. Use the argument *tableName* to specify a name for the table. If *tableName* already exists, this method overwrites it without asking for confirmation. If *tableName* is already open, this method fails.

The following table displays the structure of *tableName* is:

Field	Type	Size
ObjectName	A	128
PropertyName	A	64
PropertyType	A	48
PropertyValue	A	255

Example

{button ,AL(`OPAL_TYPE_REPORT;OPAL_METH_REENNAM;',0,"Defaultoverview",)} **Related Topics**

enumUIObjectProperties example

See the **enumUIObjectNames** example.

load method

Opens a report in the Report Design window.

Syntax

```
load ( const reportName String, [const windowStyle LongInt [ , const x LongInt, const y
LongInt, const w LongInt, const h LongInt ] ] ) Logical
```

Description

load opens *reportName* in the Report Design window. You can specify a WindowStyles constant (or combination of constants) in *windowStyle*. You can also specify the window's size and position (in twips). Arguments *x* and *y* specify the position of the upper-left corner, and arguments *w* and *h* specify the window's width and height, respectively. This method supports only saved reports (.RSL), and not delivered reports (.RDL).

Compare this method to **open**, which opens a report in the Report window.

Note

- You might need to follow a call to **open**, **load**, **design**, or **run** with a **sleep**. For more information, see the **sleep** method in the System type.

Example

{button ,AL(`OPAL_TYPE_REPORT;OPAL_METH_REDESIGN;OPAL_METH_RERUN;OPAL_METH_REOPEN;',0,"Defaultoverview",)} Related Topics

load example

In the following example, the **pushButton** method for the *loadOrders* button loads the ORDERS.RSL report in the Report Design window. This code creates a text box in the page header, and writes a string to the text box.

```
; loadOrders::pushButton
method pushButton(var eventInfo Event)
var
  ordersRep Report
  pageTitle UIObject
endVar
if ordersRep.load("Orders.rsl") then
  ; assume report has room in the page header for a text box
  pageTitle.create(TextTool, 1440*3, 720, 1440*2, 360, ordersRep)
  pageTitle.Name = "NewTitleText"
  pageTitle.Text = "Orders Report " + String(time())
  pageTitle.Color = LightBlue
  pageTitle.Visible = True
  ordersRep.run()
endIf
endMethod
```

moveToPage method

Displays the specified page of a report.

Syntax

```
moveToPage ( const pageNumber SmallInt ) Logical
```

Description

moveToPage displays the page of a report specified in *pageNumber*. This method doesn't make the report active. To make the report active, follow **moveToPage** with **bringToTop** (for more information about **bringToTop**, see the Form type).

Example

{button ,AL(`OPAL_TYPE_REPORT;OPAL_METH_FOBRTT;OPAL_METH_RECURRENTPAGE;',0,"Defaultoverview",)} Related Topics

moveToPage example

See the **currentPage** example.

open method

Opens a report.

Syntax

```
1. open ( const reportName String [, windowStyle LongInt ] ) Logical

2. open ( const reportName String, const windowStyle LongInt, const x SmallInt, const y
SmallInt, const w SmallInt, const h SmallInt ) Logical

3. open ( const openInfo ReportOpenInfo ) Logical
```

Description

open displays the report specified in *reportName*. Optional arguments specify the location of the report's upper-left corner (*x* and *y*), its width and height (*w* and *h*), and its style (*windowStyle*).

The value of *windowStyle* must be one of the WindowStyles constants. You can specify more than one window style by adding the constants. The following code opens a report window that has horizontal and vertical scroll bars:

```
salesReport.open("sales.rsl", WinStyleDefault + WinStyleHScroll + WinStyleVScroll)
```

Syntax 3 allows you to specify form settings from *openInfo*, a predefined record of type ReportOpenInfo. A ReportOpenInfo record is an instance of the Record Type, and has the following structure:

```
x, y, w, h          LongInt  ;size and position of report
name                String   ;name of report to open (preView)
masterTable         String   ;master table name
queryString         String   ;run this query (actual query string)
restartOptions      SmallInt ;one of the ReportPrintRestart constants
SQLString           String   ;run this SQL query (actual query string)
winStyle            LongInt  ;one of the WindowStyle constants
```

The MasterTable field can also be the name of a SQL file that produces an Answer table.

ReportOpenInfo now has a new field called SQLString, which can be used to specify an SQL statement to execute.

To rebind a report to a newly-created SQL statement, save the SQL statement to a file and specify the filename in ReportPrintInfo.MasterTable or ReportOpenInfo.MasterTable.

Note

- You might need to follow a call to **open**, **load**, **design**, or **run** with a **sleep**. For more information, see the **sleep** procedure in the System type.

Example

{button ,AL(`OPAL_TYPE_REPORT;OPAL_METH_RECLOS;OPAL_METH_REDESIGN;OPAL_METH_REPRINT;OPAL_METH_RELOAD;',0,"Defaultoverview",)} Related Topics

open example

In the following example, the **pushButton** method for *openSmall* opens the ORDERS.RSL report and minimizes it by supplying the window style constant WinStyleMinimize:

```
; openSmall::pushButton
method pushButton(var eventInfo Event)
var
  ordersRep Report
endVar
ordersRep.open("Orders.rsl", WinStyleMinimize)   ; open Orders Report minimized
endMethod
```

print method

Prints a report.

Syntax

```
1. print ( ) Logical
2. print ( const reportName String, const reportPrintRestart SmallInt ) Logical
3. print (const ri ReportPrintInfo ) Logical
```

Description

print prints a report. Syntax 1 instructs Corel Paradox to open the Print dialog box for the current report, which allows the user to specify print settings. Syntax 2 allows you to specify a report name in *reportName* and set restart options in *reportPrintRestart*. The value of *reportPrintRestart* must be one of the ReportPrintRestart constants. Syntax 3 lets you set print settings with a ReportPrintInfo record. The predefined ReportPrintInfo records, which are of the Record Type, have the following structure:

Field name	Type	Description
endPage	LongInt	Specifies the last page in a range (defaults to the last page of the report)
makeCopies	Logical	Specifies whether copies are made by Corel Paradox or the printer. If set to True, Corel Paradox make copies; otherwise, the printer makes copies (defaults to True). The value is ignored if the printer cannot print multiple copies.
masterTable	String	Specifies the name of the master table for the report
name	String	Specifies the name of a report to run (if one is not already running)
nCopies	SmallInt	Specifies the number of copies (defaults to one)
orient	SmallInt	Specifies the page orientation. Use one of the three ReportOrientation Constants: Landscape, Portrait, or the windows default.
pageIncrement	SmallInt	Specifies the page increment for multi-pass printing (defaults to one)
panelOptions	SmallInt	Specifies how to handle overflow pages. Use one of the ReportPrintPanel constants (defaults to PrintClipToWidth)
printBackwards	Logical	Specifies whether to print forward (from first page to last page) or backward (from last page to first page). If set to False, Corel Paradox prints forward; otherwise it prints backwards (defaults to False).
queryString	String	Specifies a QBE string to execute
restartOptions	SmallInt	Specifies what to do when data changes while printing a report. Use one of the ReportPrintRestart constants (defaults to PrintReturn)
SQLString	String	Specifies a SQL query string to execute
startPage	LongInt	Specifies the first page of a range (defaults to one)
startPageNum	LongInt	Specifies the page number to print on the first page of the report. Incremented for subsequent pages (defaults to one)
xOffset	LongInt	Specifies the horizontal page offset (defaults to zero)
yOffset	LongInt	Specifies the vertical page offset (defaults to zero)

Example

print example

The following example uses Syntax 3 to print using a ReportPrintInfo record. To print using Syntax 1, see the **attach** example.

```
; printWithRecord::pushButton
method pushButton(var eventInfo Event)
var
  stockRep  Report
  repInfo   ReportPrintInfo
endVar
; first, set up the repInfo record
repInfo.nCopies = 2
repInfo.makeCopies = True
repInfo.name = "Stock"
stockRep.print(repInfo)
endMethod
```

run method

Switches a report from the Report Design window to the Report window.

Syntax

```
run ( ) Logical
```

Description

run switches a report from the Report Design window to the View Data window. This method works only with saved reports (.RSL), and not with delivered reports (.RDL).

Use **design** to switch from the View Data window to the design window.

Note

- You might need to follow a call to **open**, **load**, **design**, or **run** with a **sleep**. For more information, see the **sleep** procedure in the System type.

Example

{button ,AL(`OPAL_TYPE_REPORT;OPAL_METH_REDESIGN;OPAL_METH_RELOAD;',0,"Defaultoverview",)}

Related Topics

run example

See the **load** example.

setMenu method

Associates a menu with a report.

Syntax

```
setMenu ( const menuVar Menu )
```

Description

setMenu associates the menu specified in *menuVar* with a report. This method performs the same function as the Menu type **show**, and adds the following:

- When the report gets focus, Corel Paradox displays the associated menu.
- Actions resulting from choices from that menu are sent to that report.

Note

- When you build a custom menu for a report, use MenuCommands constants (e.g., MenuFilePrint) to assign ID values to menu items. Because reports do not have menuAction methods for handling menu choices, these ID values are the only values that a report can respond to.

Example

{button ,AL(`OPAL_TYPE_REPORT;OPAL_METH_FOSETMENU;OPAL_METH_MUSHOW;OPAL_METH_POATEX;;',0,"Defaultoverview",)} Related Topics

setMenu example

The following example is a script that opens a report, builds a simple menu and then uses **setMenu** to assign the menu to the report:

```
method run(var eventInfo Event)
   var
      reOrders   Report
      muOrderRpt   Menu
      puRptFile   PopUpMenu
   endVar


; Build a menu for the report.
   reOrders.open("orders")


; Setting the StandardMenu property to False
; (either in ObjectPAL code or interactively)
; can reduce flicker when changing menus.
   reOrders.StandardMenu = False


; IMPORTANT: When you build a custom menu for a report,
; use MenuCommands constants (like MenuFilePrint) to assign
; ID values to menu items. These are the only values a report
; can respond to, because (unlike a form) a report has no
; menuAction method you can modify to handle menu choices.


   puRptFile.addText("&Print Report", MenuEnabled, MenuFilePrint)
   puRptFile.addText("&Exit", MenuEnabled, MenuFileExit)
   muOrderRpt.addPopUp("&File", puRptFile)
   reOrders.setMenu(muOrderRpt)
endMethod
```

Script type

Script type includes methods for manipulating scripts and the code they contain from within an ObjectPAL method or procedure.
The Script type includes several derived methods from the Form type.

Methods for the Script type

Form	**Script**
deliver	**attach**
enumSource	**create**
enumSourceToFile	**load**
formReturn	**methodEdit**
isCompileWithDebug	**run**
methodDelete	
methodGet	
methodSet	
save	
setCompileWithDebug	

{button ,AL(`OPAL_TYPE_SCRIPT;INTRO_SCRIPTS;',0,"Defaultoverview",)} **Related Topics**
Print related ObjectPAL methods and examples

attach method

Associates a Script variable with the active script.

Syntax

```
attach ( ) Logical
```

Description

attach associates a Script variable with the active script. Because this method must be called in code attached to the script itself, the script must be running. This means that **attach** allows a running script to create a handle to itself. Since ObjectPAL can't return Script variables or pass them as arguments, you must only use the handle within the method that created it. **attach** can be used with **enumSource** or **enumSourceToFile** to create a script that enumerates its own code.

This method returns True if it succeeds; otherwise, it returns False.

Example

{button ,AL(`OPAL_TYPE_SCRIPT;OPAL_METH_SCCREATE;OPAL_METH_SCLOAD;',0,"Defaultoverview",)}

Related Topics

attach example

The following example uses **attach** to create a script that enumerates its source to a text file. The code is attached to the script's built-in **run** method, which executes when you run the script.

```
method run(var eventInfo Event)
  var
    s Script
  endVar
  s.attach()
  s.enumSourceToFile("script.src", Yes)
endMethod
```

create method

Creates a script.

Syntax

create ( ) Logical

Description

create creates an empty script but *does not* display an Editor window. Use **methodSet** to add code to the script.

Example

{button ,AL(`OPAL_TYPE_SCRIPT;OPAL_METH_SCATTACH;OPAL_METH_SCLOAD;OPAL_METH_SCRUN;',0,"Defaultoverview",)} Related Topics

create example

The following example uses the **pushButton** method for a button named *editScript* to create a script named MSG. The code then calls **methodSet** to attach code to its built-in **run** method, calls **save** to save the script as NewMsg, and calls **run** to execute it. Corel Paradox automatically appends the .SSL extension.

```
; editScript::pushButton
method pushButton(var eventInfo Event)
  var
    theScript    Script
    stMsg       String
  endVar


stMsg =
"method newMsg()
  msgInfo(\"New message\", \"New message\")
endMethod"


  theScript.create()
  theScript.methodSet("run", stMsg)
  theScript.save("NewMsg") ; Saves script as NEWMSG.SSL.
  theScript.run()          ; Calls the script's built-in run method.
endMethod
```

load method

Loads a script into system memory.

Syntax

```
load ( const scriptName String ) Logical
```

Description

load loads the script specified in *scriptName* into system memory, but does not display an Editor window. If you don't specify a path or an alias in *scriptName* Corel Paradox looks for the script in the working directory. This method returns True if it succeeds; otherwise, it returns False.

Example

**{button ,AL(`OPAL_TYPE_SCRIPT;OPAL_METH_SCATTACH;OPAL_METH_SCCREATE;',0,"Defaultoverview",)
} Related Topics**

load example

The following example uses the **pushButton** method for a button named *editScript* to load the script named MSG. MSG must have been created and saved previously. The code then uses **methodSet** to add a custom method, calls **save** to save the script, and calls **run** to execute it.

```
; editScript::pushButton
method pushButton(var eventInfo Event)
  var
    theScript    Script
    stMsg       String
  endVar


stMsg =
"method newMsg()
  msgInfo(\"New message\", \"New message\")
endMethod"


  if theScript.load("msg") then
    theScript.methodSet("newMsg", stMsg)
    theScript.save()
    theScript.run() ; Executes the script's built-in run method.
  else
    errorShow("Couldn't load the script.")
  endIf
endMethod
```

methodEdit method

Opens a script's method in an Editor window.

Syntax

```
methodEdit (const methodName String) Logical
```

Description

methodEdit opens the method specified by *methodName* in an Editor window. If you specify a method that doesn't exist, **methodEdit** will create it for you. **methodEdit** fails if you try to open a method that is running.

Example

{button ,AL(`OPAL_TYPE_SCRIPT;OPAL_METH_FOMETHODSET;OPAL_METH_FOMETHODGET;OPAL_METH_FOMETHODDELETE;;',0,"Defaultoverview",)} Related Topics

methodEdit example

The following example opens the script's **testMethod** method in an editor window:

```
method pushButton(var eventInfo Event)
var
    MyScript  script
endvar
MyScript.load("update.ssl")
MyScript.methodEdit("testMethod")
endMethod
```

run method

Runs a script.

Syntax

run ( ) AnyType

Description

run runs a script by calling the script's built-in **run** method. **run** performs the same operation as the System procedure **play**. To return a value from a script, you must call **formReturn** from within the script.

Example

{button ,AL(`OPAL_TYPE_SCRIPT;OPAL_METH_SCLOAD;OPAL_METH_SYPLAY;;',0,"Defaultoverview",)}

Related Topics

run example

The following example runs a script and makes it return a value. The following code is attached to a button in a form. It runs a script and displays the returned value in a dialog box.

```
method pushButton(var eventInfo Event)
   var
      scTest      Script
      atRetVal   AnyType
   endVar


   scTest.load("test")
   atRetVal = scTest.run()
   atRetVal.view()
endMethod
```

The following code is attached to a script's built-in **run** method. It assigns a value to a variable and returns the value to the form.

```
method run(var eventInfo Event)
   var
      atNow   AnyType
   endVar
   atNow = time()
   formReturn(atNow)
endMethod
```

Session type

A Session object represents a channel to the database engine. When you launch a Corel Paradox application one session opens by default. You can use ObjectPAL to open additional sessions from within an application. Only the default session can be managed using Corel Paradox interactively. You must manage other sessions using ObjectPAL.

Locks set by ObjectPAL interact as peers with locks set interactively in the same session.

Methods for the Session type

addAlias
addPassword
addProjectAlias
advancedWildcardsInLocate
blankAsZero
close
enumAliasLoginInfo
enumAliasNames
enumDatabaseTables
enumDriverCapabilities
enumDriverInfo
enumDriverNames
enumDriverTopics
enumEngineInfo
enumFolder
enumOpenDatabases
enumUsers
getAliasPath
getAliasProperty
getNetUserName
ignoreCaseInLocate
isAdvancedWildcardsInLocate
isAssigned
isBlankZero
isIgnoreCaseInLocate
loadProjectAliases
lock
open
removeAlias
removeAllPasswords
removePassword
removeProjectAlias
retryPeriod
saveCFG
saveProjectAliases
setAliasPassword
setAliasPath
setAliasProperty
setRetryPeriod
unLock

Print related ObjectPAL methods and examples

addAlias method/procedure

Adds a public alias to a session.

Syntax

```
1. addAlias ( const aliasName String, const type String, const path String ) Logical
2. addAlias ( const aliasName String, const type String, const params DynArray[ ] String ) Logical
3. addAlias ( const aliasName String, const existingAlias String ) Logical
```

Description

addAlias adds public alias a to a session. To add a project alias, use **addProjectAlias**.

In Syntax 1, specify the alias name in *aliasName*, its (Standard) in *type*, and its full DOS path in *path*.

In Syntax 2, specify the alias name in *aliasName*, the SQL alias type (Interbase, Oracle, Sybase, or Informix) in *type*, and the parameters in *params*.

Syntax 3 copies an alias from *existingAlias* to *aliasName*.

An alias added using **addAlias** is known only to the session for which it is defined, and exists only until the session is closed. Use **saveCFG** to save public aliases in a file. By default, public aliases are stored in IDAPI.CFG. They are available from any working directory and visible to any application that uses Borland Database Engine (BDE).

Examples

{button ,AL(`OPAL_TYPE_SESSION;OPAL_METH_SSADDPROJECTALIAS;OPAL_METH_SSGALPATH;OPAL_METH_SSENUMALIASNAMES;OPAL_METH_SSSAVECFG;',0,"Defaultoverview",)} Related Topics

addAlias Method examples

Example1	Adding an alias to the active session
Example2	Adding an Oracle type alias
Example3	Copying an existing alias

addAlias example 1

The following example adds an alias to the active session and supplies the new alias to the **open** method defined for the Database type. This code is attached to the built-in **open** method for the *pageOne* page:

```
; pageOne::open
method open(var eventInfo Event)
var
  custInfo Database
endVar


; add the CustomerInfo alias to the active session
addAlias("CustomerInfo", "Standard", "D:\\Corel\\Suite8\\Paradox\\tables\\custdata")


; now use the alias to specify which database to open
custInfo.open("CustomerInfo") ; opens the CustomerInfo database


endMethod
```

addAlias example 2

The following example adds an Oracle type alias to the active session and supplies the new alias to the **open** method defined for the Database type. This code is attached to the built-in **open** method for the *pgeOne* page:

```
; pgeOne::open
method open(var eventInfo Event)
   var
      tv      TableView
      SQLdb        Database
      AliasInfo    DynArray[]  String
   endVar


   AliasInfo["SERVER NAME"]       = "Server1"
   AliasInfo["USER NAME"]            = "guest"
   AliasInfo["OPEN MODE"]            = "READ/WRITE"
   AliasInfo["SCHEMA CACHE SIZE"]    = "8"
   AliasInfo["NET PROTOCOL"]         = "SPX/IPX"
   AliasInfo["LANGDRIVER"]           = ""
   AliasInfo["SQLQRYMODE"]           = ""
   AliasInfo["PASSWORD"]             = "guest"


   addAlias("Guest_Account", "Oracle", AliasInfo)
   SQLdb.open("Guest_Account", AliasInfo)
   tv.open(":Guest_Account:mprestwood.customer")
endMethod
```

addAlias example 3

The following example adds an alias to the active session by copying the existing *work* alias to the a new alias named *NewAlias:*

```
; btnCopyWork::pushButton
method pushButton(var eventInfo Event)
   addAlias("NewAlias", "work")
endMethod
```

addPassword method/procedure

Defines a password allowing access to a protected table.

Syntax

```
addPassword ( const password String )
```

Description

addPassword provides a Corel Paradox session the password specified in *password*. Subsequent attempts to access a table protected by that password are not challenged.

The argument *password* represents an owner password or an auxiliary password. Auxiliary passwords generally confer less comprehensive rights than owner passwords. Because *password* is case-sensitive, a table protected with Sesame won't open for SESAME.

Passwords added using this method are valid only for the session for which they are defined, and exist only until the session is closed. Defining a password does not affect the state of tables (e.g., an open table remains open).

Access to tables opened before the password is presented is controlled by previously defined passwords. For example, if a table was opened using an auxiliary password, the access rights to that table do not change when the owner password is defined. To confer owner rights to a previously-opened table, close the table and present the owner password, and then reopen the table.

Use **removePassword** to restore protection to tables.

Note

- Passwords apply to Corel Paradox tables only and cannot exceed 31 characters.

Example

{button ,AL(`OPAL_TYPE_SESSION;OPAL_METH_SSRMPAS;OPAL_METH_SSRMALLPAS;',0,"Defaultoverview",)} Related Topics

addPassword example

The following example acquires a user's password, and defines it for the active session:

```
; getAddPass::pushButton
method pushButton(var eventInfo Event)
var
  newPass String
endVar
; assume that the variable ses is global, and has been
; opened by another method
if ses.isAssigned() then
  newPass.view("Enter Password (up to 31 characters) to Add.")
  ses.addPassword(newPass)
else
  msgStop("Help!","Session variable is not Assigned!")
endIf
endMethod
```

addProjectAlias method/procedure

Adds a project alias to a session.

Syntax

```
1. addProjectAlias ( const aliasName String, const type String, const path String ) Logical
2. addProjectAlias ( const aliasName String, const type String, const params DynArray[ ] String ) Logical
3. addProjectAlias ( const aliasName String, const existingAlias String ) Logical
```

Description

addProjectAlias adds a project alias to a session. Use **addAlias** to add a public alias.

In Syntax 1, specify the alias name in *aliasName*, its (Standard) in *type*, and its full DOS path in *path*.

In Syntax 2, specify the alias name in *aliasName*, the SQL alias type (Interbase, Oracle, Sybase, or Informix) in *type*, and the parameters in *params*.

Syntax 3 copies an alias from *existingAlias* to *aliasName*.

An alias added using **addProjectAlias** is known only to the project for which it is defined, and exists only until the working directory is changed. Use **saveProjectAliases** to save project aliases in a file.

When :WORK: is set (e.g., at startup) or changed (interactively or using ObjectPAL), Corel Paradox discards all current project aliases and loads those project aliases that are specific to the new working directory. Public aliases remain active and available and if a project alias has the same name as a public alias, Corel Paradox does not load the project alias. By default, Corel Paradox reads project aliases from :WORK:PDOXWORK.CFG. You can use **loadProjectAliases** to specify a different file.

Examples

{button ,AL(`OPAL_TYPE_SESSION;OPAL_METH_SSAALIAS;OPAL_METH_SSLOADPROJECTALIASES;OPAL_METH_SSREMOVEPROJECTALIAS;OPAL_METH_SSSAVEPROJECTALIASES;',0,"Defaultoverview",)} Related Topics

addProjectAlias examples

Example1 Adding an alias to a active project

Example2 Adding an Oracle type alias

Example3 Copying an existing alias

addProjectAlias example 1

The following example adds an alias to the active project and supplies the new alias to the **open** method defined for the Database type. This code is attached to the built-in **open** method for the *pageOne* page.

```
; pageOne::open
method open(var eventInfo Event)
var
  custInfo Database
endVar


; add the CustomerInfo alias to the project
addProjectAlias("CustomerInfo", "Standard", "D:\\Corel\\Suite8\\Paradox\\tables\\custdata")


; now use the alias specify the database to open
custInfo.open("CustomerInfo") ; opens the CustomerInfo database


endMethod
```

addProjectAlias example 2

The following example adds an Oracle type alias to the active project and supplies the new alias to the **open** method defined for the Database type. This code is attached to the built-in **open** method for the *pgeOne* page.

```
; pgeOne::open
method open(var eventInfo Event)
   var
      tv      TableView
      SQLdb        Database
      AliasInfo    DynArray[]  String
   endVar


   AliasInfo["SERVER NAME"]       = "Server1"
   AliasInfo["USER NAME"]            = "guest"
   AliasInfo["OPEN MODE"]            = "READ/WRITE"
   AliasInfo["SCHEMA CACHE SIZE"]    = "8"
   AliasInfo["NET PROTOCOL"]         = "SPX/IPX"
   AliasInfo["LANGDRIVER"]           = ""
   AliasInfo["SQLQRYMODE"]           = ""
   AliasInfo["PASSWORD"]             = "guest"


   addProjectAlias("Guest_Account", "Oracle", AliasInfo)
   SQLdb.open("Guest_Account", AliasInfo)
   tv.open(":Guest_Account:mprestwood.customer")
endMethod
```

addProjectAlias example 3

The following example adds an alias to the active session by copying the existing *work* alias to the new alias *NewAlias:*

```
; btnCopyWork::pushButton
method pushButton(var eventInfo Event)
   addProjectAlias("NewAlias", "work")
endMethod
```

advancedWildcardsInLocate procedure

Specifies whether the active session can use advanced wildcards in locate operations.

Syntax

```
advancedWildcardsInLocate ( [ const yesNo Logical ] )
```

Description

advancedWildcardsInLocate specifies whether the active session can use advanced wildcards found in pattern strings during locate operations. If *yesNo* is set to Yes (default), pattern strings used in locate operations can contain advanced wildcard characters. If *yesNo* is set to No, pattern strings in locate operations cannot contain advanced wildcards.

Example

{button ,AL(`OPAL_TYPE_SESSION;OPAL_METH_SSISADVANCEDWILDCARDSINLOCATE;',0,"Defaultoverview",)} Related Topics

advancedWildcardsInLocate example

The following example calls **advancedWildcardsInLocate** to determine whether advanced wildcards can be used in a locate operation. The code then calls **locatePattern** to use an advanced wildcard pattern.

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
  tc TCursor
  thisSession Session
endVar


if tc.open("Orders.db") then


  ; if advanced wild cards can't be used in patterns
  if NOT isAdvancedWildcardsInLocate() then
    ; specify that this session can use advanced
    ; pattern characters in subsequent locate operations
    advancedWildcardsInLocate(Yes)
  endIf


  if tc.locatePattern("Ship VIA", "[^UPS]") then
    msgInfo("Order Number", tc."Order No")
  else
    msgStop("Error", "Can't find record")
  endIf
else
  msgStop("Error", "Can't open Orders table.")
endIf


endMethod
```

blankAsZero method/procedure

Specifies whether to assign blank numeric fields a value of 0 in calculations.

Syntax

```
blankAsZero ( const yesNo Logical )
```

Description

blankAsZero specifies whether to assign blank numeric fields a value of 0 in calculations. If *yesNo* is set to Yes, blanks are treated as zeros. If *yesNo* is set to No blank numeric fields remain empty.

Calculations affected by **blankAsZero** include:

- calculated fields in forms and reports
- calculations in queries
- column calculations that involve the number of fields or the number of non-blank fields (e.g., those performed with **cCount**, **cAverage**, and others)

You can also use **isBlankZero** to test the state, and **blankAsZero** to set it.

Example

{button ,AL(`OPAL_TYPE_SESSION;OPAL_METH_SSISBLANKZERO;',0,"Defaultoverview",)} Related Topics

blankAsZero example

The following example sets **blankAsZero** to True so that a call to the **cAverage** method assigns blank field values a value of 0.

```
; getAvgPmt::pushButton
method pushButton(var eventInfo Event)
var
  tc TCursor
endVar


if tc.open("Orders.db") then
  if not isBlankZero() then
    blankAsZero(True)
  endIf

  msgInfo("Average Amount Paid", tc.cAverage("Amount Paid"))

else
  msgStop("Error", "Can't open Orders table.")
endIf

endMethod
```

close method

Closes a session.

Syntax

close ( ) Logical

Description

close ends a session by closing the channel to the database engine. **close** frees one user count, and leaves the Session variable unassigned.

Example

{button ,AL(`OPAL_TYPE_SESSION;OPAL_METH_SSOPEN;',0,"Defaultoverview",)} **Related Topics**

close example

The following example assumes that the variable *ses* is assigned to an open session. This example closes the session:

```
; closeSession::pushButton
method pushButton(var eventInfo Event)
; assume that the variable ses is global, and has been
; opened by another method
if ses.isAssigned() then
  if ses.close() then
    msgInfo("We have TouchDown","Session close Successful.")
  else
    msgStop("Crash and Burn","Session close Unsuccessful.")
  endIf
else
  msgStop("Help!","Session variable is not Assigned! Who am I?")
endIf
endMethod
```

enumAliasLoginInfo method

Writes server alias data to a table.

Syntax

```
enumAliasLoginInfo ( const tableName String, const aliasName String ) Logical
```

Description

enumAliasLoginInfo writes information about the server alias specified in *aliasName* to the Corel Paradox table specified in *tableName*. This method returns True if successful; otherwise, it returns False.

enumAliasLoginInfo operates on aliases that are stored in IDAPI.CFG and on new aliases opened and stored in system memory. This method fails if the table specified in *tableName* is already open.

enumAliasLoginInfo applies only to remote databases. Standard (Corel Paradox or dBASE) databases are not affected by this method.

The following table displays the structure of the resulting *tableName* table:

Field	Type	Description
DBName	A32*	Specifies the database name
Property	A32*	Specifies the property name (e.g., OPEN MODE, NET PROTOCOL, SERVER NAME, and USER NAME)
PropertyValue	A82	Specifies the property value

Example

{button ,AL(`OPAL_TYPE_SESSION;OPAL_METH_SSENUMALIASNAMES;',0,"Defaultoverview",)} Related Topics

enumAliasLoginInfo example

The following example calls **enumAliasLoginInfo** to write alias data about an alias to a Corel Paradox table. The code then searches the table to test whether the OPEN MODE property for the alias is set to READ/WRITE. If OPEN MODE is set to READ/WRITE, the code calls a custom procedure named **doSomething** to continue processing; otherwise, the code displays information about properties and property values in a modal dialog box.

```
method pushButton(var eventInfo Event)

   var
      db                Database
      aliasInfoTC       TCursor
      aliasName,
      infoTableName,
      fieldName1,
      fieldName2,
      propName,
      propVal           String
      propValDA         DynArray[] AnyType
   endVar

   ; initialize variables
   aliasName = "itchy"
   infoTableName = "dbAlias.db"
   fieldName1 = "Property"
   fieldName2 = "PropertyValue"
   propName  = "OPEN MODE"
   propVal  = "READ/WRITE"



   ; open database, get alias info
   if db.open(aliasName) then
      if enumAliasLoginInfo(infoTableName, aliasName) then
         aliasInfoTC.open(infoTableName)

         ; search for info of interest
         if aliasInfoTC.locate(fieldName1, propName) then

            ; compare expected and actual values
            if aliasInfoTC.(fieldName2) <> propVal then

               ; inform user if values don't match
               propValDA["Property:"] = aliasInfoTC.(fieldName1)
               propValDA["Expected value:"] = propVal
               propValDA["Actual value:"] = aliasInfoTC.(fieldName2)
               propValDA.view("Property mismatch")
               return
            endIf

         else
            errorShow("Property not found.")
            return
         endIf
      else
         errorShow("Can't write to table: " + infoTableName)
         return
      endIf
   else
      errorShow("Couldn't open " + aliasName)
      return
   endIf

   doSomething() ; if property values are OK, continue processing

endMethod
```

enumAliasNames method/procedure

Lists the database aliases available to a session.

Syntax

```
1. enumAliasNames ( const tableName String [ , const LoginInfoTableName String ] ) Logical

2. enumAliasNames ( var aliasNames Array[ ] String ) Logical
```

Description

enumAliasNames lists the database aliases available to a session.

Syntax 1 creates a Corel Paradox table *tableName*. If *tableName* already exists, this method overwrites it without asking for confirmation. If *tableName* is open, this method fails. You can also include an alias or path in *tableName*. If an alias or path is not specified, Corel Paradox creates the table in the working directory.

The following table displays the structure of *tableName*:

Field Name	Type	Description
DBName	A32*	Specifies the database alias name
DBType	A32	Specifies the driver type
DBPath	A82	Specifies the alias path

If you include the optional argument *LoginInfoTableName*, Corel Paradox also writes login data to the table, just as if you had called **enumAliasLoginInfo**.

The structure of the resulting table is

Field	Type	Description
DBName	A32*	Specifies the database name
Property	A32*	Specifies the property name (e.g., OPEN MODE, NET PROTOCOL, SERVER NAME, and USER NAME)
PropertyValue	A82	Specifies the property value

Syntax 2 assigns the database names to items in an array named *aliasNames* that you declare and pass as an argument.

Example

{button ,AL(`OPAL_TYPE_SESSION;OPAL_METH_SSAALIAS;OPAL_METH_SSGALPATH;',0,"Defaultoverview",)} Related Topics

enumAliasNames example

In the following example, the **pushButton** method for *getAliasButton* writes the alias names for the active session to an array. If the array does not contain the name of a specified alias, **addAlias** adds it to the session.

```
; getAliasButton::pushButton
method pushButton(var eventInfo Event)
   var
      stAliasName,
      stAliasPath   String
      arAliasNames   Array[] String
   endVar


   stAliasName = "NewCust"
   stAliasPath = "g:\\netdata\\newcust"


   enumAliasNames(arAliasNames)   ; List names to an array.
   if arAliasNames.contains(stAliasName) then
         return
   else
         addAlias(stAliasName, "STANDARD", stAliasPath)
   endIf
endMethod
```

enumDatabaseTables method/procedure

Lists the files in a database.

Syntax

```
1. enumDatabaseTables ( const tableName String, const databaseName String, const fileSpec String )

2. enumDatabaseTables ( var tableNames Array[ ] String, const databaseName String, const fileSpec String )
```

Description

enumDatabaseTables lists the files in a database specified by *databaseName*, where *databaseName* is an alias known to the session. *fileSpec* specifies a DOS file specification that can include the wildcard *.

Syntax 1 creates a Corel Paradox table named *tableName*. If *tableName* already exists, **enumDatabaseTables** overwrites it without asking for confirmation. You can also include an alias or path in *tableName*. If an alias or path is not specified, Corel Paradox creates *tableName* in the working directory.

The structure of the table is

Field	Type	Description
DBName	A32*	Specifies the database alias
TableName	A32*	Specifies the table name (or the name of another file, depending on the file specification)

Syntax 2 assigns the table names to items in an array *tableNames* that you pass as an argument.

Example

{button ,AL(`OPAL_TYPE_SESSION;OPAL_METH_SSENDRCAP;',0,"Defaultoverview",)} Related Topics

enumDatabaseTables example

The following example lists the Corel Paradox and dBASE tables (and any other file whose extension is DB followed by 0 or 1 characters) in the private directory. This code uses **enumDatabaseTables** as a procedure and works in the active session.

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
  dbName,
  fileSpec,
  tbName    String
  tv1       TableView
endVar


; Init variables.
dbName   = ":PRIV:"
fileSpec = "*.db" ; Lists <filename>.db
tbName   = "TabList"


enumDatabaseTables(tbName, dbName, fileSpec)
tv1.open(tbName)  ; Open the created table.
endMethod
```

enumDriverCapabilities procedure

Lists the capabilities of the current driver.

Syntax

enumDriverCapabilities (const ***drvCapName*** String, const ***tblCapName*** String, const ***fldCapName*** String [, const ***inxCapName*** String] **)** Logical

Description

enumDriverCapabilities creates three Corel Paradox tables that list the capabilities of the current driver. If these tables already exist, Corel Paradox overwrites them without asking for confirmation. You can also include an alias or path in the specified table names. If an alias or path is not specified, Corel Paradox creates the tables in the working directory.

Each supported table type is described by a record. Driver capabilities are written to a table named *drvCapName* which has the following structure:

Field	Type	Description
DriverType	A32*	Specifies the driver name (e.g., dBASE)
Description	A32	Describes the driver (e.g., dBASE driver)
Category	A32	Specifies the driver category
DB	A4	Specifies whether the driver supports a true database concept
DBType	A32	Specifies the database type to be used (e.g., STANDARD)
MultiUser	A4	Specifies whether the driver supports multi-user access
ReadWrite	A4	Specifies whether the driver is read-write
Transactions	A4	Specifies whether the driver supports transactions
PassThruSQL	A4	Specifies whether the driver supports pass-through SQL
Login	A4	Specifies whether the driver requires an explicit login (for example, to access a SQL server)
CreateDb	A4	Specifies whether the driver can create a database
DeleteDb	A4	Specifies whether the driver can delete a database
CreateTable	A4	Specifies whether the driver can create a table
DeleteTable	A4	Specifies whether the driver can delete a table
MultiPasswords	A4	Specifies whether the driver supports multiple passwords

Table capabilities are written to a table named *tblCapName* which has the following structure:

Field	Type	Description
DriverType	A32*	Specifies the table type (e.g., dBASE)
TableType	A32*	Describes the table type (e.g., PDOX 5.0)
Format	A32*	Specifies the table format (e.g., CLUSTERED)
ReadWrite	A4	Specifies whether the user can read from and write to the table
Create	A4	Specifies whether the user can create a table of this type
Restructure	A4	Specifies whether the user can restructure a table of this type
ValChecks	A4	Specifies whether the user can specify validity checks for a table of this type
Security	A4	Specifies whether the table can be password-protected
RefInt	A4	Specifies whether the table can participate in a referential integrity relationship
PrimaryKey	A4	Specifies whether the table supports primary keys
Indexing	A4	Specifies whether the table can have other (secondary) indexes
NoFieldType	A6	Specifies the number of physical field types supported
MaxRecSize	A6	Specifies the maximum record size (in bytes)
MaxFlds	A6	Specifies the maximum number of fields per record

Field capabilities are written to a table named *fldCapName*, which has the following structure:

Field	Type	Description
DriverType	A32*	Specifies the driver type (e.g., dBASE)
TableType	A32*	Specifies the table type (e.g., PDOX 5.0)
Format	A32*	Specifies the table format (e.g., CLUSTERED)
FieldType	A32*	Specifies the field type
Description	A32	Specifies the field type (e.g., Long integer)
NativeType	A6	Specifies the numeric value of native field type (e.g., 266)
XType	A6	Specifies the numeric value of translated field type (e.g., 3)
XSubType	A6	Specifies the numeric value of translated field subtype (e.g., 3)
MaxUnits1	A6	Specifies the maximum places to the left of the decimal point (or number of characters) (e.g., 240)
MaxUnits2	A6	Specifies the maximum places to the right of the decimal point (e.g., 19)
Size	A6	Specifies the field size (e.g., 8)
Required	A4	Specifies whether the field is a required field
Default	A4	Specifies whether the field has a specified default value
Min	A4	Specifies whether the field has a specified minimum value
Max	A4	Specifies whether the field has a specified maximum value
RefInt	A4	Specifies whether the field is part of a referential integrity relationship
Other	A4	Reserved
Key	A4	Specifies whether the field can be part of an index (keyed)
Multi	A4	Specifies whether the driver supports more than one of these fields per record
MinUnits1	A6	Specifies the minimum places to the left of the decimal point (or number of characters) (e.g., 240)
MinUnits2	A6	Specifies the minimum places to the right of the decimal point (e.g., 19)
Createable	A4	Specifies whether the driver can create a table using this field type

If you include an optional argument named *inxCapName* , index capabilities are written to the table specified in *inxCapName*. *inxCapName* has the following structure:

Field	Type	Description
DriverType	A32*	Specifies the driver type (e.g., dBASE)
TableType	A32*	Specifies the table type (e.g., PDOX 5.0)
Format	A32*	Specifies the table format (e.g., CLUSTERED)
Name	A32*	Specifies an internal name describing the type of index (e.g., SECONDARY) to correspond with the description in the Description field
Format1	A32*	Specifies the index format (e.g., BTREE)
Description	A32	Describes the index (e.g., Non-maintained Secondary index)
Composite	A4	Specifies whether the index supports composite keys
Primary	A4	Specifies whether the index is a primary index
Unique	A4	Specifies whether the index is a unique index
keyDescending	A4	Specifies whether the whole key can be descending
fldDescending	A4	Specifies whether the index is field level descending
Maintained	A4	Specifies whether the index is a maintained index
Subset	A4	Specifies whether the index is a subset index
KeyExp	A4	Specifies whether the index is an expression index
CaseInsensitive	A4	Specifies whether the index is insensitive to case

Example

{button ,AL(`OPAL_TYPE_SESSION;OPAL_METH_SSENDRINF;OPAL_METH_SSENDRNAM;OPAL_METH_SSENDRTOP;',0,"Defaultoverview",)} **Related Topics**

Categories for enumDriverCapabilities (Session type)

Value	Description
File	File-based (Corel Paradox, dBASE)
SQL Server	SQL-based server
Other Server	A server that is not file or SQL-based

Field types for enumDriverCapabilities (Session type)

The following tables display the field types for Corel Paradox and dBASE tables:

Corel Paradox field type	Return value
Alpha	ALPHA
Autoincrement	AUTOINCREMENT
BCD	BCD
Binary	BINARY
Bytes	BYTES
Date	DATE
FmtMemo	FMTMEMO
Graphic	GRAPHIC
Logical	LOGICAL
LongInt	LONG
Memo	MEMO
Money	MONEY
Number	NUMBER
OLE	OLE
Short	SHORT
Time	TIME
TimeStamp	TIMESTAMP

dBASE field type	Return value
BINARY	BINARY
CHAR	CHARACTER
DATE	DATE
FLOAT	FLOAT
LOGICAL	LOGICAL
MEMO	MEMO
NUMBER	NUMERIC
OLE	OLE

enumDriverCapabilities example

In the following example, the *describeDriver* button creates and views three tables that describe the engine driver:

```
; describeDriver::pushButton
method pushButton(var eventInfo Event)
var
  tv1, tv2, tv3  TableView
endVar
enumDriverCapabilities("dbcap", "tblcap", "fldcap")
tv1.open("dbcap")
tv2.open("tblcap")
tv3.open("fldcap")
endMethod
```

enumDriverInfo procedure

Lists information about available drivers.

Syntax

```
enumDriverInfo ( const tableName String )
```

Description

enumDriverInfo lists information about available driver types in a table named *tableName*. If *tableName* already exists, Corel Paradox overwrites it without asking for confirmation. You can also include an alias or path in *tableName*. If an alias or path is not specified, Corel Paradox creates *tableName* in the working directory.

The following table displays the structure of*tableName*:

Field	Type	Description
DriverType	A32*	Specifies the driver type or name (e.g., COREL PARADOX)
Topic	A32*	Specifies the driver function (e.g., TABLE CREATE)
Property	A32*	Specifies the property of corresponding driver function (e.g., BLOCK SIZE)
PropertyValue	A68	Specifies the value of corresponding property (e.g., 2048)

Example

{button ,AL(`OPAL_TYPE_SESSION;OPAL_METH_SSENDRCAP;OPAL_METH_SSENDRNAM;OPAL_METH_SSENDRTOP;',0,"Defaultoverview",)} **Related Topics**

enumDriverInfo example

The following example enumerates driver information to a table named *DriveInf* and displays the results:

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
  tv1 TableView
endVar
; create and view the DriveInf table
enumDriverInfo("Driveinf")
tv1.open("DriveInf")
endMethod
```

enumDriverNames method/procedure

Creates a Corel Paradox table listing the names of available drivers.

Syntax

```
enumDriverNames ( const tableName String )
```

Description

enumDriverNames writes the available driver names to *tableName*. If *tableName* already exists, Corel Paradox overwrites it without asking for confirmation. You can also include an alias or path in *tableName*. If an alias or path is not specified, Corel Paradox creates *tableName* in the working directory.

The structure of the table is DriverType, A32*.

Example

{button ,AL(`OPAL_TYPE_SESSION;OPAL_METH_SSENDRCAP;OPAL_METH_SSENDRINF;OPAL_METH_SSENDRTOP;',0,"Defaultoverview",)} Related Topics

enumDriverNames example

The following example enumerates available driver names to a table named *DrivName* and displays the results:

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
  tv1  TableView
endVar
; create and view the DrivName table
enumDriverNames("DrivName")
tv1.open("DrivName")
endMethod
```

enumDriverTopics procedure

Lists the topics currently available for each driver type.

Syntax

```
enumDriverTopics ( const tableName String )
```

Description

enumDriverTopics writes the driver topics available for each driver type to a table named *tableName*. If *tableName* already exists, Corel Paradox overwrites it without asking for confirmation. You can also include an alias or path in *tableName*. If an alias or path is not specified, Corel Paradox creates *tableName* in the working directory.

The following table displays the structure of*tableName*:

Field name	Type	Description
DriverType	A32*	Specifies the driver type or name (e.g., COREL PARADOX)
Topic	A32*	Specifies the driver function For Corel Paradox and dBASE tables, the topics are INIT and TABLE CREATE

Example

 Related Topics

enumDriverTopics example

The following example enumerates available driver topics to a table named *DrivTop* and displays the results:

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
  tv1  TableView
endVar
; create and view the DrivTop table
enumDriverTopics("drivtop")
tv1.open("drivtop")
endMethod
```

enumEngineInfo procedure

Creates a Corel Paradox table listing the current Borland Database Engine (BDE) engine properties.

Syntax

```
enumEngineInfo ( const tableName String )
```

Description

enumEngineInfo creates a Corel Paradox table that describes the contents of the BDE System Information dialog box. Each setting name and value is written to a record in a table named *tableName*. If *tableName* already exists, Corel Paradox overwrites it without asking confirmation. You can also include an alias or path in *tableName*. If an alias or path is not specified, Corel Paradox creates *tableName* in the working directory.

The following table displays the structure of*tableName*:

Field name	Type	Description
Property	A32*	Specifies the engine property
PropertyValue	A68	Specifies the value of corresponding property

Example

{button ,AL(`OPAL_TYPE_SESSION;OPAL_METH_SSENDRINF;',0,"Defaultoverview",)} **Related Topics**

enumEngineInfo example

The following example enumerates engine information to a table named *EngInf* and displays the results:

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
  tv1  TableView
endVar
enumEngineInfo("EngInf")
tv1.open("EngInf")
endMethod
```

Properties for enumEngineInfo

Engine property	Description
LANGDRIVER	Specifies the name of language driver (e.g., ASCII)
LANGDRVDIR	Specifies the language driver folder
LOCAL SHARE	Specifies whether the Local Share is active
MAXBUFSIZE	Specifies the maximum buffer size (in bytes)
MAXFILEHANDLES	Specifies the maximum number of file handles
MINBUFSIZE	Specifies the minimum buffer size (in bytes)
NET DIR	Specifies the path to NET folder
NET TYPE	Specifies the network type
SYSFLAGS	Specifies the number of system flags
VERSION	Specifies the BDE version number

enumFolder procedure

Lists the names of files in a folder or project.

Syntax

```
1. enumFolder ( const tableName String [ , const fileSpec String ] ) Logical
2. enumFolder ( var result Array[ ] String [ , const fileSpec String ] ) Logical
```

Description

enumFolder lists the names of files in a folder or project. By default, a project includes all the objects in :WORK: and :PRIV:. You can also add references to objects in other directories.

Syntax 1 creates a Corel Paradox table named *tableName*. If *tableName* already exists, this method overwrites it without asking for confirmation. You can also include an alias or path in *tableName*. If an alias or path is not specified, Corel Paradox creates *tableName* in the working directory.

Syntax 2 lists the files in an array named *result* which you must declare and pass as an argument. For each file, the array lists the filename (and extension, if one exists), and includes the path if the file is not in the working directory.

You can list files using a particular extension using an optional argument named *fileSpec*. For example, to list all forms in a file, specify .FSL in*fileSpec*.

The structure of the table created by Syntax 1 is

Field	Type	Description
Name	A128	Specifies the filename (and extension, if one exists). Includes the path if the file is not in :WORK:.
LocalName	A68	Specifies the filename without extension. Includes the path if the file is not in :WORK:.
IsReference	A4	Specifies whether the filename refers to a file in a directory other than :WORK:
IsPrivate	A4	Specifies whether the filename refers to a file in :PRIV:
IsTemp	A4	Reserved
Position	A10	Reserved

Example

{button ,AL(`OPAL_TYPE_SESSION;OPAL_METH_SSENDBTAB;OPAL_METH_SSLOADPROJECTALIASES;',0,"Defaultoverview",)} **Related Topics**

enumFolder example

In the following example, the method prompts the user to type a file specification (e.g., *.FSL). The file specification entered is then used by **enumFolder** to create a table listing the files that match the specification.

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
  filespec String
  tv1      TableView
endVar
filespec.view("Enter filename specification")
enumFolder("PartCat", filespec)
message("Table lists files that match your specification.")
tv1.open("PartCat")
endMethod
```

enumOpenDatabases method/procedure

Lists the open databases.

Syntax

```
1. enumOpenDatabases ( const tableName String ) Logical
2. enumOpenDatabases ( var tableNames Array[ ] String ) Logical
```

Description

enumOpenDatabases lists the databases open in the active session.

Syntax 1 creates a Corel Paradox table named *tableName*. If *tableName* already exists, this method overwrites it without asking for confirmation. You can also include an alias or path in *tableName*. If an alias or path is not specified, Corel Paradox creates *tableName* in the working directory.

The following table displays the structure of the resulting table:

Field name	Type	Description
DBName	A32*	Specifies the database alias name
DBType	A32	Specifies the database driver type
ShareMode	A32	Specifies the database share mode
OpenMode	A32	Specifies the database open mode

Syntax 2 writes the data to an array *tableNames* that you declare and pass as an argument.

Example

{button ,AL(`OPAL_TYPE_SESSION;OPAL_METH_SSENDBTAB;',0,"Defaultoverview",)} Related Topics

enumOpenDatabases example

In the following example, **enumOpenDatabases** creates a table named *OPENDB.DB*, and then displays the table.

```
; btnOpenDB :: pushButton
method pushButton(var eventInfo Event)
   var
      tv   TableView
   endVar


   enumOpenDatabases("OPENDB.DB")
   tv.open("OPENDB.DB")
endMethod
```

enumUsers procedure

Creates a Corel Paradox table listing all known users with an open channel to the Borland Database Engine (BDE) engine.

Syntax

```
1. enumUsers ( const tableName String ) LongInt
2. enumUsers ( var userNames Array[ ] String ) LongInt
```

Description

enumUsers creates a list of all users with an open path to the BDE database engine.

Syntax 1 creates a table named *tableName* that lists all users with an open path to BDE. If *tableName* already exists, Corel Paradox overwrites it without asking for confirmation. You can also include an alias or path in *tableName.* If an alias or path is not specified, Corel Paradox creates *tableName* in the working directory.

The following table displays the structure of the resulting table:

Field Name	Type	Description
UserName	A15	Specifies the network user name
NetSession	N	Specifies the network session number
ProductClass	N	Specifies the user's product class ID number
SerialNumber	A22	Specifies the serial number (version 1.0 only)

Syntax 2 lists the network names of users who currently have an open path to BDE in an array. You must declare the array before calling this procedure.

Example

{button ,AL(`OPAL_TYPE_SESSION;OPAL_METH_SSGETNETUSERNAME;',0,"Defaultoverview",)} **Related Topics**

enumUsers example

The following example writes information about current users to a table named *Users* and displays the table:

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
  tv1 TableView
endVar
  enumUsers("users")
  tv1.open("users")
endMethod
```

getAliasPath method/procedure

Returns the path for a specified alias.

Syntax

```
getAliasPath ( const aliasName String ) String
```

Description

getAliasPath returns the path for an alias named *aliasName*.

Example

{button ,AL(`OPAL_TYPE_SESSION;OPAL_METH_SSAALIAS;OPAL_METH_SSSETALIASPATH;',0,"Defaultoverview",)} Related Topics

getAliasPath example

The following example prompts the user for an alias name and displays the corresponding path:

```
; getShowPath::pushButton
method pushButton(var eventInfo Event)
   var
      stPrompt,
      stAliasName,
      stCurrentPath,
      stMyPath      String
   endVar


   stPrompt    = "Enter an Alias Name."
   stAliasName = stPrompt
   stMyPath    = "d:\\Corel\\Suite8\\Paradox\\data"


   stAliasName.view(stPrompt)    ; prompt for an alias name
   if stAliasName = stPrompt then
         return ; User didn't click the OK button.
   else
         stCurrentPath = getAliasPath(stAliasName)  ; get the path
   endIf


   if stCurrentPath = stMyPath then
         return
   else
         setAliasPath(stAliasName, stMyPath)
   endIf
endMethod
```

getAliasProperty method

Returns the property value for a specified server alias.

Syntax

```
getAliasProperty ( const aliasName String, const property String ) String
```

Description

getAliasProperty returns a string representing the property value specified by *property* for the server alias specified in *aliasName*. If the property is not valid for the alias, this method returns an error. **getAliasProperty** operates on aliases stored in IDAPI.CFG and on new aliases that have been opened and stored in system memory.

This method only applies to remote databases, and not to standard (Corel Paradox or dBASE) databases.

Example

{button ,AL(`OPAL_TYPE_SESSION;OPAL_METH_SSSETALIASPROPERTY;',0,"Defaultoverview",)} Related Topics

getAliasProperty example

The following example uses **getAliasProperty** to retrieve the value of the OPEN MODE property. This code compares the returned (actual) value with the expected value. If the returned and expected values match, the code calls a custom procedure named doSomething to continue processing. If the returned and expected values do not match, the code informs the user of a property mismatch and calls **setAliasProperty** to set the property to the expected value.

```
method pushButton(var eventInfo Event)


   var
      db                Database
      aliasName,
      propName,
      expectedPropVal,
      actualPropVal     String
      propValDA         DynArray[] AnyType
   endVar


   ; initialize variables
   aliasName = "itchy"
   propName = "OPEN MODE"
   expectedPropVal = "READ/WRITE"


   if db.open(aliasName) then


      ; get property value and compare with expected value
      actualPropVal = getAliasProperty(aliasName, propName)
      if actualPropVal = expectedPropVal then
         doSomething() ; continue processing
         return
      else


         ; inform the user if there's a mismatch
         propValDA["Property name"] = propName
         propValDA["Expected value"] = expectedPropVal
         propValDA["Actual value"] = actualPropVal
         propValDA.view("Property mismatch:")


         ; let user decide what to do
         if msgQuestion("Set property value?",
            "Set "+propName+" to " + expectedPropVal + "?") = "Yes" then

            ; set property to expected value and continue processing
          if setAliasProperty(aliasName, propName, expectedPropVal) then
               doSomething() ; Continue processing
               return
          else
               errorShow("Couldn't set property value.",
                         "Operation canceled.")
               return
          endIf


         else
            msgInfo("Operation canceled.", "Property not set.")
            return
         endIf


      endIf


   else
      msgStop(aliasName, "Couldn't open database.")
      return
   endIf
```

```
endMethod
```

getNetUserName method/procedure

Returns the name of the current network user.

Syntax

getNetUserName ( ) String

Description

getNetUserName returns the name of the current network user.

Example

{button ,AL(`OPAL_TYPE_SESSION;OPAL_METH_SSENUMUSERS;',0,"Defaultoverview",)} **Related Topics**

getNetUserName example

The following example displays the current user's network name in a dialog box:

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
msgInfo("Who Am I?", getNetUserName())
endMethod
```

ignoreCaseInLocate procedure

Specifies whether to ignore case-sensitivity in locate operations.

Syntax

```
ignoreCaseInLocate ( [ const yesNo Logical ] )
```

Description

ignoreCaseInLocate specifies whether the active session ignores case-sensitivity during locate operations. If an optional argument named *yesNo* is set to Yes or omitted, all subsequent locate operations ignore case in string comparisons. If *yesNo* is set to No, locate operations will respect case.

Example

{button ,AL(`OPAL_TYPE_SESSION;OPAL_METH_SSISIGNORECASEINLOCATE;',0,"Defaultoverview",)}

Related Topics

ignoreCaseInLocate example

The following example calls **ignoreCaseInLocate** to prepare for a call to the **locate** method:

```
; findName::pushButton
method pushButton(var eventInfo Event)
var
  tc                TCursor
   loIgnoreCase   Logical
endVar

if tc.open("Customer.db") then

   loIgnoreCase = isIgnoreCaseInLocate() ; Get user's setting.

  if loIgnoreCase then

    ; locate values based on value as entered
    ; (do not ignore case in string compares)
    ignoreCaseInLocate(No)
  endIf

  ; search for case-sensitive MacAnaly in Name field
  if tc.locate("Name", "MacAnaly") then
    tc.edit()
    tc.Name = "Macanaly"
    tc.endEdit()
  else
    message("Couldn't find MacAnaly...")
  endIf

   ignoreCaseinLocate(loIgnoreCase) ; Restore user's setting.

else
  msgStop("Error", "Can't open Customer table.")
endIf

endMethod
```

isAdvancedWildcardsInLocate procedure

Reports whether the active session is using advanced wildcards during locate operations.

Syntax

```
isAdvancedWildcardsInLocate ( ) Logical
```

Description

isAdvancedWildcardsInLocate reports whether the active session is using advanced wildcards during locate operations that include pattern strings.

Example

{button ,AL(`OPAL_TYPE_SESSION;OPAL_METH_SSADVANCEDWILDCARDSINLOCATE;',0,"Defaultoverview",)} Related Topics

isAdvancedWildcardsInLocate example

The following example calls **advancedWildcardsInLocate** to specify that advanced wild cards can be used in a locate operation. The code the calls to **locatePattern**, which uses an advanced wildcard pattern.

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
  tc TCursor
  thisSession Session
endVar


if tc.open("Orders.db") then


  ; if advanced wild cards can't be used in patterns
  if NOT isAdvancedWildcardsInLocate() then
    ; specify that this session can use advanced
    ; pattern characters in subsequent locate operations
    advancedWildcardsInLocate(Yes)
  endIf


  if tc.locatePattern("Ship VIA", "[^UPS]") then
    msgInfo("Order Number", tc."Order No")
  else
    msgStop("Error", "Can't find record")
  endIf
else
  msgStop("Error", "Can't open Orders table.")
endIf


endMethod
```

isAssigned method

Reports whether a Session variable is assigned.

Syntax

isAssigned ( ) Logical

Description

isAssigned reports whether a Session variable is assigned.

Example

{button ,AL(`OPAL_TYPE_SESSION;OPAL_METH_SSOPEN;OPAL_METH_SSCLOSE;',0,"Defaultoverview",)}

Related Topics

isAssigned example

See the **close** example.

isBlankZero method/procedure

Reports whether blank values are treated as zero in calculations.

Syntax

isBlankZero ( ) Logical

Description

isBlankZero returns True if blank fields are treated as zero in calculations**,** or as filled fields in counting calculation (e.g., **cCount**). If blank fields are treated as blanks or are being ignored in calculations and counts, **isBlankZero** returns False. Use **blankAsZero** to change this setting.

Example

{button ,AL(`OPAL_TYPE_SESSION;OPAL_METH_SSBLANKASZERO;',0,"Defaultoverview",)} Related Topics

isBlankZero example

See the **blankAsZero** example.

isIgnoreCaseInLocate procedure

Reports whether the active session ignores case-sensitivity in locate operations.

Syntax

isIgnoreCaseInLocate ( ) Logical

Description

isIgnoreCaseInLocate reports whether the active session ignores case-sensitivity during locate operations.

Example

{button ,AL(`OPAL_TYPE_SESSION;OPAL_METH_SSIGNORECASEINLOCATE;',0,"Defaultoverview",)}

Related Topics

isIgnoreCaseInLocate example

See the **ignoreCaseInLocate** example.

loadProjectAliases procedure

Loads project alias specifications.

Syntax

```
loadProjectAliases ( const cfgFileName String ) Logical
```

Description

loadProjectAliases loads project alias specifications from the file specified in *cfgFileName*. If *cfgFileName* does not specify a path, Corel Paradox searches for the file in the working directory. Corel Paradox automatically reads project aliases from :WORK:PDOXWORK.CFG. This method lets you specify a different file.

When :WORK: is set (e.g., at startup) or changed (interactively or through ObjectPAL), Corel Paradox discards all current project aliases and loads those project aliases that are specific to the new working directory. Public aliases remain active and available. If a project alias has the same name as a public alias, Corel Paradox does not load the project alias. This method returns True if it succeeds; otherwise, it returns False.

Example

{button ,AL(`OPAL_TYPE_SESSION;OPAL_METH_SSAALIAS;OPAL_METH_SSADDPROJECTALIAS;OPAL_METH_SSREMOVEPROJECTALIAS;OPAL_METH_SSSAVEPROJECTALIASES;OPAL_METH_FSSETWORKINGDIR;',0,"Defaultoverview",)} **Related Topics**

loadProjectAliases example

The following example loads the project aliases in the **open** method of the form's first page. This code reads a list of custom aliases from C:\COREL\SUITE8\PARADOX\CUSTOM.CFG instead of from the Corel Paradox default configuration file.

```
;pge1 :: open
method open(var eventInfo Event)
   loadProjectAliases("C:\\COREL\UITE8\\PARADOX\\CUSTOM.CFG")
endMethod
```

lock procedure

Locks one or more tables.

Syntax

```
lock ( const table { Table|TCursor|String },  const lockType String [ , const table { Table|
TCursor| String },  const lockType String ] * ) Logical
```

Description

lock locks one or more of the tables specified in comma-separated pairs of tables and lock types. You can use a TCursor or a Table to specify a table. You can mix TCursor and Table variables in the list.

The following *lockType* values are listed in order of decreasing strength and increasing concurrency:

String value	Description
Full	Specifies whether the active session has exclusive access to the table. Cannot be used with dBASE tables.
Write	Specifies whether the active session can write to and read from the table. No other session can place a write lock or a read lock on the table.
Read	Specifies whether the active session can read from the table. No other session can place a write lock, full lock, or exclusive lock on the table.

If **lock** locks all the tables in the list, it returns True; otherwise, it returns False. If **lock** can't lock all the tables, it doesn't lock any.

Example

{button ,AL(`OPAL_TYPE_SESSION;OPAL_METH_SSUNLOCK;',0,"Defaultoverview",)} **Related Topics**

lock example

The following example attempts to place a write lock on the *Orders* table and a read lock on the *Customer* table. If **lock** is able to lock both tables, the code displays data from both tables in a dialog box. The code then calls **unlock** to remove the explicit locks placed on *Customer* and *Orders*.

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
  ordTB    Table
  custTC   TCursor
  sampDB   Database
  otherSes Session
endVar


otherSes.open("other") ; Open another session
otherSes.addAlias("samples", "Standard", "c:\\Corel\\Suite8\\Paradox\\samples")
sampDB.open("samples", otherSes)


custTC.open("Customer.db", sampDB)
ordTB.attach("Orders.db", sampDB)


if lock(custTC, "Read", ordTB, "Write") then
  if custTC.locate("Name", "Unisco") then
    custNo = custTC."Customer No"
    ordTB.setIndex("Customer No")
    ordTB.setFilter(custNo, custNo)
    msgInfo(String("Total for order ", custNo),
            ordTB.cSum("Total Invoice"))
    unlock(custTC, "Read", ordTB, "Write")
  else
    msgStop("Error", "Can't find Unisco.")
  endIf
else
  errorShow()
endIf


endMethod
```

open method

Opens a session (a channel to the database engine).

Syntax

```
1. open ( ) Logical

2. open ( const sessionName String ) Logical
```

Description

open opens a session (a channel to the database engine). Calling **open** with no arguments (Syntax 1) gives you a handle to the active session; it does not exhaust a user count. When you use *sessionName* to specify a session name (Syntax 2), you open another channel to the database engine and exhaust one user count. The *sessionName* value can be any valid string.

If you open multiple sessions from the same workstation, Corel Paradox views each session as a separate user (e.g., locks set in one session block access from the other).

Example

{button ,AL(`OPAL_TYPE_SESSION;OPAL_METH_SSCLOSE;',0,"Defaultoverview",)} Related Topics

open example

The following example calls **open** to retrieve a handle to the active session, and to open a new session. The code then calls **blankAsZero** to specify how each session handles blank values in calculations. Finally, the code passes the Session variables to a custom procedure named doSomething. Because different sessions have different **blankAsZero**, the results of doSomething vary.

```
; openSession::pushButton
method pushButton(var eventInfo Event)
var
  currentSes,
  otherSes    Session
endVar


; Open sessions.
currentSes.open()
otherSes.open("other")


; Set session properties.
currentSes.blankAsZero(Yes)
otherSes.blankAsZero(No)


; Pass session handles to a custom procedure.
; Results will differ depending on settings for each session.
doSomething(currentSes)
doSomething(otherSes)


endMethod
```

removeAlias method/procedure

Removes an alias from a session.

Syntax

```
removeAlias ( const aliasName String ) Logical
```

Description

removeAlias removes the alias *aliasName* from a session. You cannot remove :WORK:, :PRIV:, or an open alias.

Example

{button ,AL(`OPAL_TYPE_SESSION;OPAL_METH_SSAALIAS;',0,"Defaultoverview",)} **Related Topics**

removeAlias example

The following example adds an alias to the active session and makes the new alias available to the **open** method defined for the Database type. When the alias is no longer needed, this code calls **removeAlias** to remove the alias from the active session.

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
  custInfo Database
endVar


; Add the CustomerInfo alias to the current session.
addAlias("CustomerInfo", "Standard", "D:\\Corel\\Suite8\\Paradox\\tables\\custdata")


; Now use the alias specify the database to open.
custInfo.open("CustomerInfo") ; Opens the CustomerInfo database.


; Do something with the opened database,
; then when the alias is no longer needed, close the
; database and remove the alias from the current session.


custInfo.close()
removeAlias("CustomerInfo")


endMethod
```

removeAllPasswords method/procedure

Removes passwords defined for a session.

Syntax

```
removeAllPasswords ( )
```

Description

removeAllPasswords removes passwords defined for a session. This method withdraws the passwords required to access protected tables, but does not remove security from tables.

Example

{button ,AL(`OPAL_TYPE_SESSION;OPAL_METH_SSAPASS;OPAL_METH_SSRMPAS;',0,"Defaultoverview",)}

Related Topics

removeAllPasswords example

The following example removes all the passwords from the session named *ses*.

```
; removePasses::pushButton
method pushButton(var eventInfo Event)
; assume that the variable ses is global, and has been
; opened by another method
if ses.isAssigned() then
  ses.removeAllPasswords()
else
  msgStop("Help!","Session variable is not Assigned!")
endIf
endMethod
```

removePassword method/procedure

Removes a password defined for a session.

Syntax

```
removePassword ( const password String )
```

Description

removePassword removes a password defined for a session. This method withdraws the password specified in the argument *password* , but does not unprotect the table. *password* is case-sensitive.

Example

{button ,AL(`OPAL_TYPE_SESSION;OPAL_METH_SSAPASS;OPAL_METH_SSRMALLPAS;',0,"Defaultoverview",)} Related Topics

removePassword example

In the following example, the *getRemovePass* button acquires a password to remove from the user and removes the password from the active session. Subsequent attempts to open tables protected by that password fail.

```
; getRemovePass::pushButton
method pushButton(var eventInfo Event)
var
  newPass string
endVar
; assume that the variable ses is global, and has been
; opened by another method
if ses.isAssigned() then
  newPass.view("Enter Password to Remove")
  ses.removePassword(newPass)
else
  msgStop("Help!", "Session variable is not Assigned!")
endIf
endMethod
```

removeProjectAlias procedure

Removes a project alias. For information about aliases, see Public and Project Aliases in the User's Guide Help.

Syntax

```
removeProjectAlias ( const alias String ) Logical
```

Description

removeProjectAlias removes the project alias specified in *alias*.

When the working directory is set (e.g., at startup) or changed (interactively or through ObjectPAL), Corel Paradox discards all current project aliases and loads those project aliases that are specific to the new working directory. Public aliases remain active and available. If a project alias has the same name as a public alias, Corel Paradox does not load the project alias. By default, Corel Paradox reads project aliases from :WORK:PDOXWORK.CFG; however, you can use **loadProjectAliases** to specify a different path and file.

Example

{button ,AL(`OPAL_TYPE_SESSION;OPAL_METH_SSAALIAS;OPAL_METH_SSADDPROJECTALIAS;OPAL_METH_SSLOADPROJECTALIASES;OPAL_METH_SSREMOVEALIAS;OPAL_METH_SSSAVEPROJECTALIASES;OPAL_METH_FSSETWORKINGDIR;',0,"Defaultoverview",)} Related Topics

removeProjectAlias example

The following example uses **addProjectAlias** in the page's built-in **arrive** method to add an alias to the current project. The code then uses **removeProjectAlias** in the page's built-in **depart** method to remove the alias.

The following code is attached to the page's built-in **arrive** method:

```
;pge1 :: arrive
method arrive(var eventInfo MoveEvent)
   ;Add the CustomerInfo alias to the project.
   addProjectAlias("CustomerInfo", "Standard", "D:\\COREL\\SUITE8\\PARADOX\\SAMPLES")
endMethod
```

The following code is attached to the page's built-in **depart** method.

```
;pge1 :: depart
method depart(var eventInfo MoveEvent)
   ;Remove the CustomerInfo alias from the project.
   if not removeProjectAlias("CustomerInfo") then
      errorShow("Could not remove project alias CustomerInfo.")
   endIf
endMethod
```

retryPeriod method/procedure

Returns the number of seconds allowed to retry an operation on a locked record or table.

Syntax

```
retryPeriod ( ) SmallInt
```

Description

retryPeriod returns the number of seconds allowed to retry an operation on a locked record or table. If the **retryPeriod** is set to 0 (default), operations are not retried.

Example

{button ,AL(`OPAL_TYPE_SESSION;OPAL_METH_SSSRETRY;',0,"Defaultoverview",)} Related Topics

retryPeriod example

The following example displays the current retry period:

```
; getShowRetry::pushButton
method pushButton(var eventInfo Event)
var
  rp smallint
endVar
; assume that the variable ses is global, and has been
; opened by another method
if ses.isAssigned() then
  rp = ses.RetryPeriod()            ; get the current retry period
  rp.view("The Retry Period is...") ; display the value
else
  msgStop("Help!","Session variable is not assigned!")
endIf
endMethod
```

saveCFG method/procedure

Saves the active session's alias information to a file.

Syntax

```
saveCFG ( const fileName String ) Logical
```

Description

saveCFG saves the BDE configuration for the active session in *fileName*. The configuration file specified by *fileName* can be loaded using the **-o** command-line option to set session information at startup.

Example

{button ,AL(`OPAL_TYPE_SESSION;OPAL_METH_SYDLGNETSYSTEM;',0,"Defaultoverview",)} Related Topics

saveCFG example

The following example saves the current BDE settings to MyConfig.cfg.

```
;// saveconfiguration::pushButton
method pushButton(var eventInfo Event)
;// saves the BDE setting to file MyConfig.cfg
saveCfg("MyConfig.cfg")
endMethod
```

saveProjectAliases procedure

Saves project alias specifications to a file. For information about aliases, see Public and project aliases in the User's Guide Help.

Syntax

```
saveProjectAliases ( [ const fileName String ] ) Logical
```

Description

saveProjectAliases saves project alias specifications to a file. You can use the optional argument *fileName* to specify a filename. If you omit *fileName*, Corel Paradox saves the alias to :WORK:PDOXWORK.CFG.

When :WORK: is set (e.g., at startup) or changed (interactively or through ObjectPAL), Corel Paradox discards all current project aliases and loads those project aliases that are specific to the new working directory. Public aliases remain active and available. If a project alias has the same name as a public alias, Corel Paradox does not load the project alias. By default, Corel Paradox reads project aliases from :WORK:PDOXWORK.CFG; however, you can use **loadProjectAliases** to specify a different path and file.

Example

{button ,AL(`OPAL_TYPE_SESSION;OPAL_METH_SSAALIAS;OPAL_METH_SSADDPROJECTALIAS;OPAL_METH_SSLOADPROJECTALIASES;OPAL_METH_SSREMOVEPROJECTALIAS;OPAL_METH_FSSETWORKINGDIR;',0,"Defaultoverview",)} **Related Topics**

saveProjectAliases example

The following example uses **saveProjectAliases** to save new project aliases to MYPROJ.CFG:

```
;pge1 :: open
method open(var eventInfo Event)
   ; Add project alias.
   addProjectAlias("MYPROJ", "Standard", "D:\\COREL\\SUITE8\\PARADOX\\\MYPROJ")


   ; Save project aliases.
   saveProjectAliases("MYPROJ.CFG")
endMethod
```

setAliasPassword method

Sets the in-memory password for a specified alias.

Syntax

```
setAliasPassword ( const aliasName, const password String ) Logical
```

Description

setAliasPassword sets the in-memory password for the alias specified in *aliasName* to the value specified in *password*. Passwords have a maximum length of 31 characters. The next time you open that alias, you do not have to supply the password.

Calling **setAliasPassword** has the same effect as defining a password interactively using the Alias Manager dialog box. **setAliasPassword** has no effect on the password stored and maintained on the server. This method returns True if successful; otherwise, it returns False.

Example

{button ,AL(`OPAL_TYPE_SESSION;OPAL_METH_SSGETALIASPROPERTY;OPAL_METH_SSSETALIASPROPERTY;',0,"Defaultoverview",)} Related Topics

setAliasPassword example

The following example calls **setAliasPassword** to define the password for a specified alias. When the call to **open** executes, this code opens the database without prompting the user for a password.

```
method pushButton(var eventInfo Event)
   var
      aliasName,
      aliasPassword   String

      db              Database
   endVar


   ; initialize variables
   aliasName = "bedrock"
   aliasPassword = "fred" ; Max length: 31 characters


   ; set alias password and open database
   if setAliasPassword(aliasName, aliasPassword) then
   db.open(aliasName) ; opens without prompting for password
   else
      errorShow("Couldn't set alias password.")
      return
   endIf

endMethod
```

setAliasPath method/procedure

Sets the path for an alias.

Syntax

```
setAliasPath ( const aliasName String, const aliasPath String ) Logical
```

Description

setAliasPath sets the path *aliasPath* for the alias *aliasName*.

Example

{button ,AL(`OPAL_TYPE_SESSION;OPAL_METH_SSGALPATH;',0,"Defaultoverview",)} Related Topics

setAliasPath example

See the **getAliasPath** example.

setAliasProperty method

Sets the value of a specified property for a specified alias.

Syntax

```
setAliasProperty ( const aliasName String, const property String, const propertyValue String )
Logical
```

Description

setAliasProperty sets the value specified in *property,* to the value specified in *propertyValue,* for the alias specified in *aliasName*. This method returns True if successful; otherwise, it returns False.

Properties that you set using this method are displayed in the Alias Manager dialog box. Since property settings are *not* automatically saved to IDAPI.CFG, you must use the Session procedure **saveCFG** to save alias properties to a file.

This method applies only to remote databases, and not to standard (Corel Paradox or dBASE) databases.

Example

{button ,AL(`OPAL_TYPE_SESSION;OPAL_METH_SSGETALIASPROPERTY;OPAL_METH_SSSETALIASPASSWORD;',0,"Defaultoverview",)} Related Topics

setAliasProperty example

See the **getAliasProperty** example.

setRetryPeriod method/procedure

Sets the number of seconds allowed to retry an action on a locked table or record.

Syntax

```
setRetryPeriod ( const period SmallInt ) Logical
```

Description

setRetryPeriod specifies the number of seconds to allowed retry an action on a locked table or record. If you set **setRetryPeriod** to 0, actions are not retried.

Example

{button ,AL(`OPAL_TYPE_SESSION;OPAL_METH_SSRETRY;',0,"Defaultoverview",)} **Related Topics**

setRetryPeriod example

The following example prompts the user to specify a retry period and sets the session's retry period to that value:

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
  rp Smallint
endVar
; assume that the variable ses is global, and has been
; opened by another method
if ses.isAssigned() then
  rp = ses.retryPeriod()
  rp.view("Enter retry period")  ; get a retry period from user
  ses.setRetryPeriod(rp)         ; set the session's retry period
else
  msgStop("Help!","Session variable is not assigned!")
endIf
endMethod
```

unlock procedure

Unlocks one or more tables.

Syntax

```
unlock ( const table { Table|TCursor|String } ,
         const lockType String [ , const table { Table|TCursor|String } ,
         const lockType String ] * ) Logical
```

Description

unlock unlocks one or more of the tables specified in a comma-separated list of tables and lock types.

unlock removes locks explicitly placed by a particular user or application but does not affect locks placed automatically by Corel Paradox. The *lockType* value must be one of the following: Exclusive, Write, Read, or Full. Read and Full apply only to Corel Paradox tables.

If one **unlock** attempt fails, previous locks are not restored—the tables remain unlocked. You don't have to specify a session in which to use this method, because session data is set when you **open** a TCursor or **attach** to a Table.

To ensure maximum concurrent availability of tables, unlock tables when the lock is no longer required. When you lock a table twice, you must unlock it twice. You can use the **lockStatus** method (defined for the TCursor and UIObject types) to determine how many explicit locks you have placed on a table. If you try to unlock a table that isn't locked or cannot be unlocked, **unlock** returns False.

Example

{button ,AL(`OPAL_TYPE_SESSION;OPAL_METH_SSLOCK;OPAL_METH_TCLSTA;OPAL_METH_UILOCKSTATUS;',0,"Defaultoverview",)} **Related Topics**

unlock example

See the **lock** example.

SmallInt type

SmallInt values are small integers that can be represented by a short series of digits. A SmallInt variable occupies 2 bytes of storage.

ObjectPAL converts SmallInt values to range from -32,768 to 32,767. If you attempt to assign a value outside of this range to a SmallInt variable, an error occurs.

```
var
   x, y, z SmallInt
endVar


x = 32767 ; The upper limit value for a SmallInt variable.
y = 1
z = x + y ; This statement causes an error.
```

When ObjectPAL performs an operation on SmallInt values, the result must also be a SmallInt value. To work with a boundary value (in either the positive or negative direction), convert it to a type that can accommodate it. In the following example, ObjectPAL converts a SmallInt to a LongInt before performing the addition. The result is assigned to a LongInt variable which can handle the large value.

```
var
   x, y SmallInt
   z    LongInt   ; Declare z as a LongInt so it can hold the result.
endVar


x = 32767 ; the upper limit value for a SmallInt variable
y = 1
z = LongInt(x) + y
```

Notes

- The SmallInt value -32,768 cannot be stored in a Corel Paradox table. Corel Paradox considers -32,768 to be a blank. This value can be used in calculations and stored in a dBASE table. Store such large numbers as LongInt or Number data types.
- Run-time library methods and procedures defined for the Number type also work with LongInt and SmallInt variables. The syntax is the same, and the returned value is a Number. For example, the following code returns a Number value, even though sin does not appear in the methods for the SmallInt type:

```
var
   abc LongInt
   xyz Number
endVar
abc = 43
xyz = abc.sin()
```

The SmallInt type includes several derived methods from the Number and AnyType types.

Methods for the SmallInt type

AnyType	Number	**LongInt**
blank	abs	**bitAND**
dataType	acos	**bitIsSet**
isAssigned	asin	**bitOR**
isBlank	atan	**bitXOR**
isFixedType	atan2	**int**
view	ceil	**smallInt**
	cos	
	cosh	
	exp	
	floor	
	fraction	
	fv	
	ln	
	log	
	max	

min
mod
number
numVal
pmt
pow
pow10
pv
rand
round
sin
sinh
sqrt
tan
tanh
truncate

Print related ObjectPAL methods and examples

bitAND method

Performs a bitwise AND operation on two values.

Syntax

```
bitAND ( const value SmallInt ) SmallInt
```

Description

bitAND returns the result of a bitwise AND operation on *value*. **bitAND** operates on the binary representations of two integers, comparing them one bit at a time. The truth table for **bitAND** is:

a	b	a bitAND b
0	0	0
1	0	0
0	1	0
1	1	1

Example

{button ,AL(`OPAL_TYPE_SMALLINT;OPAL_METH_SIBOR;OPAL_METH_SIBXOR;',0,"Defaultoverview",)}

Related Topics

bitAND example

In the following example, the **pushButton** method for a button named *andTwoNums* performs a bitwise AND calculation on two integers. The result is displayed in a dialog box.

```
; andTwoNums::pushButton
method pushButton(var eventInfo Event)
var
  a, b SmallInt
endVar
a = 30233   ; binary 01110110 00011001
b = 1233    ; binary 00000100 11010001
a.bitAND(b) ; binary 00000100 00010001
msgInfo("The result of 30233 bitAND 1233 is:", a.bitAND(b))
; displays 1041
endMethod
```

bitIsSet method

Reports whether a bit is 1 or 0.

Syntax

```
bitIsSet ( const value SmallInt ) Logical
```

Description

bitIsSet examines the binary representation of an integer and reports whether the **value** bit is 0 or 1. This method returns True if the bit is 1, and False if it is 0.

value is a number specified by 2^n, where *n* is an integer between 0 and 14. The exponent *n* corresponds to one position less than the position of the bit to test, counting from the right. For example, to specify the third bit from the right, use
$4(2^{(3-1)}$, which is 2^2).

Examples

{button ,AL(`OPAL_TYPE_SMALLINT;OPAL_METH_SIBAND;OPAL_METH_SIBOR;OPAL_METH_SIBXOR;',0,"Defaultoverview",)} **Related Topics**

bitIsSet method examples

Example1 Using **pushButton**

Example2 Using **bitIsSet** to display an integer as a binary number

bitIsSet example 1

The following example uses the **pushButton** method for a button named *isABitSet*, to examine the values in two unbound field objects: *whichBit* and *whatNum*. *whichBit* contains the bit position (counting from the right) of the bit test. *whatNum* contains the integer to test. The **pushButton** method uses *whichBit* to calculate the value of the position, then assigns the result to *bitNum*. The code then checks *Num* to determine whether the *bitNum* bit is set and displays the Logical result in a **msgInfo** dialog box.

```
; isABitSet::pushButton
method pushButton(var eventInfo Event)
var
  bitNum,
  Num      SmallInt
endVar
; get the bit position number from the whichBit
; field and convert to multiple of 2
bitNum = SmallInt(pow(2, whichBit - 1))
; get the number to test from the whatNum field
Num = whatNum
; is the bit for value bitNum 1 in Num?
msgInfo("Is Bit Set?", Num.bitIsSet(bitNum))
endMethod
```

bitIsSet example 2

The following example uses **bitIsSet** to display an integer as a binary number. The **pushButton** method for *showBinary* constructs a string of zeros and ones, by testing each bit of a four-byte integer. A blank is added after the 8th digit in the string for readability.

```
; showBinary::pushButton
method pushButton(var eventInfo Event)
var
  binString  String    ; to construct the binary string
  Num        SmallInt  ; number to test
  i          SmallInt  ; for loop index
endVar
if NOT whatNum.isBlank() then
  Num = whatNum                ; get the number test from whatNum
  binString = ""               ; initialize the string
  for i from 0 to 14
    if Num.bitIsSet(SmallInt(pow(2, i))) then
      binString = "1" + binString    ; add a 1 to the front of the string
    else
      binString = "0" + binString    ; add a 0 to the front of the string
    endIf
    if i = 7 then
      binSTring = " " + binString    ; add a space every 8 digits
    endIf
  endfor
  if Num < 0 then
    binString = "1" + binString      ; set the sign bit
  else
    binString = "0" + binString
  endIf
  ; show the number
  message("The binary equivalent is ", binString)
endIf
endMethod
```

bitOR method

Performs a bitwise OR operation on two values.

Syntax

```
bitOR ( const value SmallInt ) SmallInt
```

Description

bitOR performs a bitwise OR operation on *value*. **bitOR** operates on the binary representations of two integers, comparing them one bit at a time. The truth table for **bitOR** is:

a	b	a bitOR b
0	0	0
1	0	1
0	1	1
1	1	1

Example

{button ,AL(`OPAL_TYPE_SMALLINT;OPAL_METH_SIBAND;OPAL_METH_SIBXOR;',0,"Defaultoverview",)}
Related Topics

bitOR example

In the following example, the **pushButton** method for a button named *orTwoNums* performs a bitwise OR calculation on two integers. The result is displayed in a dialog box.

```
; orTwoNums::pushButton
method pushButton(var eventInfo Event)
var
  a, b SmallInt
endVar
a = 30233  ; binary 01110110 00011001
b = 1233   ; binary 00000100 11010001
a.bitOR(b) ; binary 01110110 11011001
msgInfo("30233 OR 1233", a.bitOR(b)) ; displays 30425
endMethod
```

bitXOR method

Performs a bitwise XOR operation on two values.

Syntax

```
bitXOR ( const value SmallInt ) SmallInt
```

Description

bitXOR performs a bitwise XOR (exclusive OR) operation on *value*. **bitXOR** operates on the binary representations of two integers, comparing them one bit at a time. The truth table for **bitXOR** is:

a	b	a bitXOR(b)
0	0	0
1	0	1
0	1	1
1	1	0

Example

{button ,AL(`OPAL_TYPE_SMALLINT;OPAL_METH_SIBAND;OPAL_METH_SIBOR;',0,"Defaultoverview",)}

Related Topics

bitXOR example

In the following example, the **pushButton** method for a button named *xorTwoNums* takes two integers and performs a bitwise XOR calculation on them. The result of the calculation is displayed in a dialog box.

```
; xorTwoNums::pushButton
method pushButton(var eventInfo Event)
var
  a, b SmallInt
endVar
a = 30233   ; binary 01110110 00011001
b = 1233    ; binary 00000100 11010001
a.bitXOR(b) ; binary 01110010 11001000
msgInfo("30233 XOR 1233", a.bitXOR(b)) ; displays 29384
endMethod
```

int procedure

Casts a value as an integer.

Syntax

```
int ( const value AnyType ) SmallInt
```

Description

int casts the numeric expression *value* to an integer. If *value* is of a more precise type (e.g., Number), precision is lost.

Example

{button ,AL(`OPAL_TYPE_SMALLINT;OPAL_METH_SISMIN;',0,"Defaultoverview",)} Related Topics

int example

The following example assigns a number to *nn*, views the value of *nn* in a dialog box and displays *nn* as an integer. This code is attached to the **pushButton** method for the *showInt* button:

```
; showInt::pushButton
method pushButton(var eventInfo Event)
var
   nn Number
endVar
nn = 123.12
view(nn)                              ; displays 123.12
msgInfo("nn as Integer", int(nn))   ; displays 123
endMethod
```

smallInt procedure

Casts a value as a small integer.

Syntax

```
smallInt ( const value AnyType ) SmallInt
```

Description

smallInt casts the numeric expression *value* to a SmallInt. If *value* is of a more precise type (e.g., Number), precision is lost.

Example

{button ,AL(`OPAL_TYPE_SMALLINT;OPAL_METH_SIINT;',0,"Defaultoverview",)} Related Topics

smallInt example

The following example assigns a number to *x*, casts *x* to SmallInt, and assigns the result to *s*. The decimal precision of *x* is lost when it is cast as a SmallInt.

```
; convertToInt::pushButton
method pushButton(var eventInfo Event)
var
  x Number
  s SmallInt
endVar
x = 12.34             ; give x a value
x.view()              ; view x, title of dialog will be "Number"
s = SmallInt(x)       ; cast x as a LongInt and assign to s
s.view()              ; show s, note that decimal places are lost
                      ; displays 12
endMethod
```

SQL type

An ObjectPAL SQL variable represents an SQL statement. You can use ObjectPAL to create and execute SQL commands from methods in the same way that your create and execute SQL commands interactively. SQL commands can be executed from an SQL file, an SQL statement, or a string. Some queries require Corel Paradox to create temporary tables in the private directory.

Methods for the SQL type

executeSQL
getQueryRestartOptions
isAssigned
readFromFile
readFromString
setQueryRestartOptions
wantInMemoryTCursor
writeSQL

 **Print related ObjectPAL methods and examples**

executeSQL method/procedure

Executes an SQL statement.

Syntax

```
Method:
1. executeSQL ( const db Database) Logical
2. executeSQL ( const db Database, ansTbl String ) Logical
3. executeSQL ( const db Database, ansTbl Table ) Logical
4. executeSQL ( const db Database, ansTbl TCursor ) Logical

Procedure:
1. executeSQL ( const db Database, const qbeVar SQL ) Logical
2. executeSQL ( const db Database, const qbeVar SQL, ansTbl String ) Logical
3. executeSQL ( const db Database, const qbeVar SQL, ansTbl Table ) Logical
4. executeSQL ( const db Database, const qbeVar SQL, ansTbl TCursor ) Logical
```

Description

executeSQL executes a pass through SQL query created in an ObjectPAL method or procedure.

In Syntax 1 the answer table is not specified. **executeSQL** writes to ANSWER.DB in the private directory.

In Syntax 2 the answer table is specified as a string. If you do not include a file extension, the answer table is a Corel Paradox table by default.

In Syntax 3 *ansTbl* is a Table variable. *ansTbl* must be assigned and valid.

In Syntax 4 a TCursor is opened onto the answer set. The TCursor may be an in-memory table or a cursor onto the answer set.

executeSQL returns True if the query is executed on the server (even if the resulting table is empty); otherwise, it returns False.

An SQL query in ObjectPAL code begins with an SQL variable, the = sign, and the keyword SQL followed by a blank line. The code continues with the SQL statements that make up the body of the query, followed by another blank line. The query ends with the keyword **endSQL**. Because this query is not a quoted string, it can contain tilde variables.

Note

- **executeSQL** is a pass through function. The SQL statements are sent directly to the server as if by another user. SQL statements do not execute within the context of a database handle or active transaction.

Example

{button ,AL(`OPAL_TYPE_SQL;OPAL_METH_SQLREADFROMFILE;OPAL_METH_SQLREADFROMSTRING;',0,"Defaultoverview",)} Related Topics

executeSQL example

The following example prompts the user to type an item name and stores the user's input in a variable. The code then it uses the variable as a tilde variable in an SQL query and calls **executeSQL** to execute the query. The results are stored in a TCursor in system memory. If the query executes successfully, the results are passed to a custom procedure for additional processing.

```
method pushButton(var eventInfo Event)
   var
      itemNameSQL SQL
      ViewName tableview
      db database
   endVar
db.open(":aliasname:");this will open the connection to the local table or SQL table via the
alias
itemNameSQL = SQL ;stores the SQL statement into the variable
SELECT DISTINCT Field1, Field2, etc.
FROM "DatabaseName.DB"
endSQL
executeSQL(db, itemNameSQL, ":aliasname:answertablename") ;execute the SQL statement into an
answer table
view2.open(":aliasname:answertablename") ;to view the answer table
endMethod
```

isAssigned method

Reports whether an SQL variable has an assigned value.

Syntax

isAssigned ( ) Logical

Description

isAssigned returns True if an SQL variable has been assigned a value; otherwise, it returns False. **isAssigned** does not determine if the assigned SQL statement is valid.

Example

{button ,AL(`OPAL_TYPE_SQL;OPAL_METH_QUERY;',0,"Defaultoverview",)} Related Topics

isAssigned example

In the following example, the call to **isAssigned** returns True. The SQL variable *sqlVar* has been assigned a value even though the value is not a valid SQL variable.

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
  sqlVar SQL
endVar
sqlVar = SQL
     This is not a valid SQL statement
     endSQL
msgInfo("Assigned?", sqlVar.isAssigned())    ; displays True
endMethod
```

readFromFile method

Assigns the contents of an SQL file to an SQL variable.

Syntax

```
readFromFile ( const sqlFileName SQL ) Logical
```

Description

readFromFile assigns the contents of *sqlFileName* to an SQL variable. *SqlFileName* is created with **writeSQL** or interactively with the SQL Editor. Do not use the **SQL** and **endSQL** keywords. Use **executeSQL** to execute the query.

If *fileName* does not include a path or alias, **readFromFile** searches for the file in the directory associated with the specified database (or the default database, if a database is not specified). If the value of *fileName* does not include an extension, **readFromFile** assumes an extension of .SQL. To specify a filename that does not have an extension, type a period after the name. The following table lists the filenames different *fileName* values:

***fileName* value**	**SQL filename**
newcust	newcust.sql
newcust.	newcust
newcust.s	newcust.s

readFromFile returns True if it succeeds; otherwise, it returns False.

Note

- **readFromFile** is a pass through function. The SQL statements are sent directly to the server as if by another user. SQL statements do not execute within the context of a database handle or active transaction.

Example

{button ,AL(`OPAL_TYPE_SQL;OPAL_METH_SQLEXECUTESQL;OPAL_METH_SQLREADFROMSTRING;',0,"Defaultoverview",)} Related Topics

readFromFile example

The following example creates a pop-up menu listing the SQL files stored in the private directory. When the user chooses a file from the menu, this code calls **readFromFile**. **readFromFile** reads the query, assigns it to an SQL variable, executes the query, and stores the results in a TCursor. The code then passes the TCursor to a custom procedure (assumed to be defined elsewhere) for additional processing.

```
method pushButton(var eventInfo Event)
   var
      myAlias,
      aliasTableName,
      sqlFileName,
      sqlFileSpec      String
      aliasNamTC,
      answerTC         TCursor
      sqlPop           PopUpMenu
      db               Database
      sqlFS            FileSystem
      sqlFileAr        Array[] String
      sqlVar           SQL
   endVar


   ; initialize variables
   myAlias = "itchy"
   aliasTableName = ":PRIV:aliasNam.db"
   sqlFileSpec = ":PRIV:*.SQL"


   enumAliasNames(aliasTableName) ; create a table of aliases


   aliasNamTC.open(aliasTableName)
   if aliasNamTC.locate("DBName", myAlias) then
      db.open(myAlias)  ; use alias to get database handle to server
   else
      msgStop("Stop",
              "The alias " + myAlias +
              " has not been defined.")
      return ; exit the method
   endIf


   ; build a pop-up menu listing SQL files in the target directory
   if sqlFS.findFirst(sqlFileSpec) then
      sqlFS.enumFileList(sqlFileSpec, sqlFileAr)
      sqlPop.addArray(sqlFileAr)
      sqlFileName = sqlPop.show() ; variable stores user's menu choice


      ; read and execute the SQL file chosen by the user
      sqlVar.readFromFile(sqlFileName)
      if sqlVar.executeSQL(db,answerTC) then
         doSomething(answerTC) ; call custom proc to process data
      else
         errorShow("readFromFile failed")
      endIf


   else
      msgStop("File not found:", sqlFileSpec)
   endIf


endMethod
```

readFromString method

Assigns a query string to an SQL variable.

Syntax

```
readFromString ( const sqlString SQL ) Logical
```

Description

readFromString assigns the SQL query string specified in *sqlString* to an SQL variable. Do not enclose the string between the **SQL** and **endSQL** keywords. Use **executeSQL** to execute the query.

Notes

- **readFromFile** is a pass through function. The SQL statements are sent directly to the server as if by another user. SQL statements do not execute within the context of a database handle or active transaction.

Example

{button ,AL(`OPAL_TYPE_SQL;OPAL_METH_SQLEXECUTESQL;OPAL_METH_SQLREADFROMFILE;',0,"Defaultoverview",)} Related Topics

readFromString example

The following example prompts the user to type an SQL keyword and uses that keyword in an SQL string. If the user enters a valid SQL keyword and the query executes successfully, the results are stored in a TCursor and passed to a predefined custom procedure for additional processing.

```
method pushButton(var eventInfo Event)
   var
      sqlKeyword,
      promptString,
      bigOrderString   String
      aliasNamTC,
      bigOrderTC       TCursor
      db               Database
      myAlias,
      aliasTableName   String
      sqlVar           SQL
   endVar


   ; Initialize variables.
   myAlias = "itchy"
   aliasTableName = ":PRIV:aliasNam.db"
   promptString = "Enter an SQL keyword (e.g. SELECT):"


   enumAliasNames(aliasTableName)


   ; Prompt user to enter an SQL keyword.
   sqlKeyword.view("SQL Keyword")
   if sqlKeyword = promptString then
      return ; Exit method if user doesn't enter a keyword.
   endIf


   ; Use alias to open database.
   aliasNamTC.open(aliasTableName)
   if aliasNamTC.locate("DBName", myAlias) then
      db.open(myAlias)  ; Use alias to get database handle to server
   else
      msgStop("Stop", "The alias " + myAlias +
                      " has not been defined.")
      return
   endIf



   ; Combine SQL statements and String variable sqlKeyword
   ; to create an SQL string.
   bigOrderString = sqlKeyword +
                    "CustName, Order_no, Sale_date, Qty
                    FROM     Customer
                    WHERE    Qty > 1000 "


   ; Read and execute the query and process the results.
   sqlVar.readFromString(bigOrderString)
   if sqlVar.executeSQL(bigOrderTC) then
      doSomething(bigOrderTC) ; call custom proc to process data
   else
      errorShow()
   endIf


endMethod
```

wantInMemoryTCursor method

Specifies how to create a TCursor resulting from a query.

Syntax

```
wantInMemoryTCursor ( [ const yesNo Logical ] )
```

Description

wantInMemoryTCursor specifies how to create a TCursor from a query. When you execute a query to a TCursor, that TCursor points to a live query view and changes made to the TCursor affect the underlying tables. When you call **wantInMemoryTCursor** with *yesNo* set to Yes or omitted, Corel Paradox creates the TCursor in system memory, without a connection to underlying tables.

An in-memory TCursor is especially useful for performing quick what-if analyses. For example, to study the effect of giving each employee a 15 percent raise, you can query the employee data to increase all salaries by 15 percent. If you execute the query to an in-memory TCursor, you can manipulate the data there, without affecting the actual employee data.

Example

{button ,AL(`OPAL_TYPE_SQL;OPAL_METH_SQLEXECUTESQL;OPAL_METH_QUWANTINMEMORYTCURSOR;OPAL_METH_TCINSTANTIATEVIEW;OPAL_METH_TCISINMEMORYTCURSOR;OPAL_METH_TCISVIEW;',0,"Defaultoverview",)} Related Topics

wantInMemoryTCursor example

The following example uses an in-memory TCursor to study the effects of giving all employees a 15 percent raise. The code reads a predefined query from a file and uses the results in a calculation.

```
method pushButton(var eventInfo Event)
   var
      qVar            SQL
      tcRaise15       TCursor
      nuTotalPayroll  Number
      MyDB            Database
   endVar


   MyDB.open("work")
   qVar.wantInMemoryTCursor(Yes)
   qVar.readFromFile("raise15.sql")
   qVar.executeSQL(MyDB, tcRaise15)


   nuTotalPayroll = tcRaise15.cSum("Salary")
   nuTotalPayroll.view("Payroll after 15%   raise:")
endMethod
```

writeSQL method/procedure

Writes an SQL statement or an SQL string to a file.

Syntax

```
Method:
1. writeSQL ( const fileName String ) Logical

Procedure:
2. writeSQL ( const sqlString String, const fileName String ) Logical
```

Description

writeSQL writes a predefined SQL statement or SQL string to the file specified in *fileName*. If *fileName* already exists, Corel Paradox overwrites it without asking for confirmation. **writeSQL** returns True if successful; otherwise, it returns False. This method does not evaluate the SQL commands.

Syntax 1 is a method—use dot notation to specify an SQL variable (e.g., sqlVar.writeSQL("bigOrder.sql").

Syntax 2 is a procedure—use a String variable as the first argument (e.g., writeSQL(sqlString, "bigOrder.sql").

Example

{button ,AL(`OPAL_TYPE_SQL;OPAL_METH_SQLEXECUTESQL;OPAL_METH_SQLREADFROMSTRING;OPAL_METH_SQLREADFROMFILE;',0,"Defaultoverview",)} Related Topics

writeSQL example

The following example prompts the user to type a table name and stores the name in a String variable. The code then it uses the String variable as a tilde variable in an SQL statement. The call to **writeSQL** writes the SQL statement (including the expanded tilde variable) to a file. If the user types ORDERS as the table name, the resulting SQL file would contain the following statement:

```
SELECT * FROM ORDERS
```

writeSQL does not determine whether the SQL statements are valid.

```
method pushButton(var eventInfo Event)
   var
      sqlString       SQL
      userTableName,
      sqlFileName,
      promptString    String
   endVar


   ; Initialize variables.
   sqlFileName = "user001.sql"
   promptString = "Enter table name here."
   userTableName = promptString


   ; Display a view() dialog box and prompt user for input.
   userTableName.view("Select * from table:")


   ; If user enters a string, use it in a tilde variable
   ; in the following SQL query.
   if userTableName <> promptString then
      sqlString =
         SQL
            SELECT * FROM ~userTableName
         endSQL
      writeSQL(sqlString, sqlFileName) ; Write user's query to a file.
   endIf

endMethod
```

StatusEvent type

StatusEvent type methods control messages that appear in the desktop Status Bar. Using StatusEvent type methods, you can attach code to built-in event methods to determine where and why messages are displayed. You can block messages or display them in a different status area, or in another object (e.g., a field object or text file). You can also use StatusEvent type methods to specify the text to be displayed in the message.

Use the StatusReasons constants to refer to specific areas on the Status Bar.

The StatusEvent type includes several derived methods from the Event type.

Methods for the StatusEvent type

Event	**StatusEvent**
errorCode	**reason**
getTarget	**setReason**
isFirstTime	**setStatusValue**
isPreFilter	**statusValue**
isTargetSelf	
reason	
setErrorCod	
setReason	

Print related ObjectPAL methods and examples

reason method

Reports why a StatusEvent occurred.

Syntax

reason ( ) SmallInt

Description

reason returns an integer value that reports why a StatusEvent occurred. StatusEvent reasons occur each time a built-in **status** method is called. ObjectPAL uses StatusReasons constants to test the value returned by **reason**.

Example

{button ,AL(`OPAL_TYPE_STATUSEVENT;OPAL_METH_SESREA;',0,"Defaultoverview",)} Related Topics

reason example

The following example copies all the messages that are sent to the Status Bar to a field. Assume that a form contains a field named *fldStatus*. The form's built-in **status** method examines the event packet to determine the reason. If the reason is StatusWindow, the form's built-in **status** method sends the status value to a field named *fldStatus*.

```
;frm1 :: status
method status(var eventInfo StatusEvent)
if eventInfo.isPreFilter()
   then
      ; This code executes for each object on the form.
   else
      ; This code executes only for the form.
      if eventinfo.reason() = StatusWindow then
         fldStatus.Value = eventinfo.statusValue()
      endIf
endIf
endMethod
```

setReason method

Specifies a reason for generating a StatusEvent.

Syntax

```
setReason ( const reasonId SmallInt )
```

Description

setReason specifies a reason for generating a StatusEvent. StatusEvent reasons indicate which Status Bar window received the message. ObjectPAL uses StatusReasons constants to set the reason for a StatusEvent.

Example

{button ,AL(`OPAL_TYPE_STATUSEVENT;OPAL_METH_SEREAS;OPAL_METH_EVERRORCODE;OPAL_METH_EVSETERRORCODE;',0,"Defaultoverview",)} **Related Topics**

setReason example

In the following example, for StatusEvent bubbled up to the form from a field, the form's **status** method changes the reason and the content of the message. The code changes the reason to ModeWindow1, and sets the message value to the name of the object that initiated the event (the target).

```
; thisForm::status
method status(var eventInfo StatusEvent)
var
  targObj  UIObject
  nameStr  String
endVar
if eventInfo.isPreFilter()
  then
    ; code here executes for each object in form
  else
    ; code here executes just for form itself
    ; after regular message has displayed, also show
    ; field name in ModeWindow1
    eventInfo.getTarget(targObj)
    if targObj.Class = "Field" then      ; if this is a field
      nameStr = targObj.Name       ; get the field name
      eventInfo.setReason(ModeWindow1)   ; set the window
      eventInfo.setStatusValue(nameStr)  ; send the string
    endIf
endIf
endMethod
```

setStatusValue method

Specifies the text of a status message.

Syntax

```
setStatusValue ( const statusValue AnyType )
```

Description

setStatusValue specifies the text of a status message.

Example

{button ,AL(`OPAL_TYPE_STATUSEVENT;OPAL_METH_SESREA;OPAL_METH_SESTVAL;',0,"Defaultoverview",)} Related Topics

setStatusValue example

See the **setReason** example.

statusValue method

Returns the text of a status message.

Syntax

```
statusValue ( ) AnyType
```

Description

statusValue returns the text of a status message.

Examples

{button ,AL(`OPAL_TYPE_STATUSEVENT;OPAL_METH_SEREAS;OPAL_METH_SESSTAT;',0,"Defaultoverview",)} Related Topics

statusValue method examples

statusValue example 1

The following example makes the default status messages more prominent to a user by copying each message to a field on the form. This feature is controlled by the *magnifyMessage* button on the same form. The following code is attached to the **pushButton** method of the *magnifyMessage* button:

```
; magnifyMessage::pushButton
method pushButton(var eventInfo Event)
; toggle statusMessageField to visible or invisible and
; toggle label between "Magnified Messages" and "Normal Messages"
if self.LabelText = "Magnified Messages" then
  statusMessageField.Visible = True
  self.LabelText = "Normal Messages"
else
  statusMessageField.Visible = False
  self.LabelText = "Magnified Messages"
endIf
endMethod
```

The following is attached to the form's **status** method:

```
; thisForm::status
method status(var eventInfo StatusEvent)
if eventInfo.isPreFilter()
  then
    ; code here executes for each object in form
    ; write every status event to a field on the form
    if statusMessageField.Visible = True then
      if eventInfo.reason() = StatusWindow then
        statusMessageField = eventInfo.statusValue()
      endIf
    endIf
  else
    ; code here executes just for form itself


endIf
endMethod
```

statusValue example 2

In this example, code is placed in the **status** method at the form's page level and traps for a change in the Persistent Field View setting. *modeWindow3* refers to the right field of the message line and displays the current view setting (e.g., Field View, Persistent Field View or Memo View). If the field is in Persistent Field View, a custom method named **persistFldVw** is called to perform predefined actions.

```
method status(var eventInfo StatusEvent)
if eventinfo.reason() = modeWindow3 then
   if eventinfo.statusvalue()="Persist " then   ; note "Persist " is
                                                ; followed by a space
       persistFldVw()                           ; call custom method
   endIf
endIf
endMethod
```

String type

Strings store and manipulate alphanumeric data. A String variable's length is limited to the virtual memory on your computer. Strings occupy 1 byte of storage space per character. Empty stings are represented by double quotes ("")

String lengths may also be limited according to their use. For example, if you assign a String variable to an Alpha field in a Corel Paradox table, the String variable cannot exceed the width of the Alpha field.

The String type includes several derived methods from the AnyType type.

Notes

- ObjectPAL supports an alternate syntax:

```
methodName ( objVar , argument [ , argument ] )
```

methodName represents the name of the method, *objVar* is the variable representing an object, and *argument* represents one or more arguments. For example, the following statement uses the standard ObjectPAL syntax to return a lowercase version of a string:

```
theString.lower()
```

The following statement uses the alternate syntax:

```
lower(theString)
```

It's best to use standard syntax for clarity and consistency, but you can use the alternate syntax wherever it's convenient.

- Virtual memory is related to available disk space. For more information, see your Windows documentation.

Methods for the String type

AnyType
blank
dataType
isAssigned
isBlank
isFixedType
view

String
advMatch
ansiCode
breakApart
chr
chrOEM
chrToKeyName
fill
format
ignoreCaseInStringCompares
isEmpty
isIgnoreCaseInStringCompares
isSpace
keyNameToChr
keyNameToVKCode
lower
lTrim
match
oemCode
readFromClipboard
rTrim
search
searchEx
size
sizeEx
space
string
strVal
substr

toANSI
toOEM
upper
vkCodeToKeyName
writeToClipboard

Print related ObjectPAL methods and examples

advMatch method

Searches text for a specified string.

Syntax

```
advMatch ( const pattern String [ , var matchVar String ] * ) Logical
```

Description

advMatch returns True if *pattern* is found within the string; otherwise, it returns False. To specify *pattern*, use a string and the optional symbols listed in the table. By default, this method is case sensitive by default. Use the String procedure **ignoreCaseInStringCompares** to change the case-sensitivity.

advMatch assigns matched patterns to *matchVar* variables as the patterns are found. The portions of the string that match wildcard elements are assigned to the variables from left to right. Because there multiple matches might be found, the first matching substring is assigned to the first variable, the second matching substring to the second variable, and so on. If no match is found, variables are not assigned values.

If you supply *pattern* from within a method, you must use two backslashes to instruct **advMatch** to treat a special character as a literal. For example, \\(tells **advMatch** to treat the parenthesis as a literal character.

If you're trying to search for a question mark embedded in a string, you might call **advMatch** like so:

```
s = "a string?"
advMatch(s, "\?")       ; this won't work!
```

You might think that you're telling **advMatch** to search for the literal question mark. However, the compiler sees the string first and returns a syntax error because \? is not a valid escape sequence. To prevent the compiler from interpreting the backslash as the beginning of an escape sequence, precede the backslash by another backslash. This will work:

```
s = "a string?"
advMatch(s, "\\?")       ; this does work!
```

If you supply *pattern* from a field in a table or a TextStream, special **advMatch** symbols are recognized without a preceding backslash. In this case, one backslash and plus symbol (\+) yields a literal character.

Symbol	Matches
\	Include special characters (e.g., \t for Tab) as regular characters. Use two backslashes in quoted strings.
[ ]	Match the enclosed set. (e.g., [aeiou0-9] matches a, e, i, o, u, and 0 through 9)
[^]	Do not match the enclosed set. (e.g., [^aeiou0-9] match anything except a, e, i, o, u, and 0 through 9)
( )	Grouping
^	Beginning of string
$	End of string
..	Match anything
@	Match any single character
*	Zero or more of the preceding character or expression
+	One or more of the preceding character or expression
?	None or one of the preceding character or expression
|	OR operation

Example

{button ,AL(`OPAL_TYPE_STRING;OPAL_METH_STMAT;OPAL_METH_STSEARCH;OPAL_METH_TSAMAT;',0,"Defaultoverview",)} Related Topics

advMatch example

The following example demonstrates **advMatch** functionality:

```
method pushButton(var eventInfo Event)
var
  w, x, y, z      String
  l               Logical
endVar


l = advMatch("this is", "s")
l.view()
  ; returns True (different from match)


l = advMatch("this is", "^s")
l.view()
  ; returns False, because it requires s to be at the beginning of the line


l = advMatch("this is", "S")
l.view()
  ; returns False, it is case sensitive.


l = advMatch("this is", "[sS]")
l.view()
  ; returns True, because [sS] specifies any in this set


l = advMatch("this is", "[a-z]")
l.view()
  ; returns True, because [a-z] specifies any in this set of a through z


l = advMatch("this is", "[a-c]")
l.view()
  ; returns False, because [a-c] specifies any in this set of a through c
  ; and "this is" does not contain a, b, or c


l = advMatch("this is", "[a-cs]")
l.view()
  ; returns True, because [a-cc] specifies any in this set of a through c
  ; or s and "this is" does contain s
  ; note that [a-c, s] would specify any in the set of a through c,
  ; a comma, a space, or an s


l = advMatch("this is", "(@)s", x)
l.view()
x.view()
  ; returns True, x = "i" because the "()" operators specify a group,
  ; unlike match, advMatch places only those things that you group
  ; in the variables


l = advMatch("this is a test", "((t@@s)|(t@s))|(@s)", w, x, y, z)
l.view()   ; returns True, and
w.view()   ; "this", the result of the first set of parentheses,
           ; that is, for the entire expression ((t@@s)|(t@s))
           ; also, "this" was matched before "test"
x.view()   ; also "this", for the result of the second set of
           ; parentheses, (t@@s)
y.view()   ; the result of (t@s), blank, because the t@@s
           ; satisfied the expression ((t@@s)|(t@s))
z.view()   ; also blank, because the expression ((t@@s)|(t@s)) satisfied
           ; the entire pattern ((t@@s)|(t@s))|(@s)
; NOTE: Match variables are matched to groups in the order of occurrence,
;       not in the order of precedence: The first group starting from
;       the left

is assigned to the first variable.
```

```
l = advMatch("this is so", "(..)is(..)", x, y)
l.view()
x.view()
y.view()
  ; returns True, x = "this", y = " so"


l = advMatch("this is so", "[a-c]|[f-l]s" )
l.view()
  ; returns True, because an s is preceded by either a through
  ; c or f through l


l = advMatch("this as so", "[a-c]|[t-z]s" )
l.view()
  ; returns True, because an s is preceded by either a through
  ; c or t through z

endMethod
```

ansiCode procedure

Returns the ANSI code of a one-character string.

Syntax

```
ansiCode ( const char String ) SmallInt
```

Description

ansiCode returns the ANSI code of a one-character string. The returned value is an integer between 1 and 255.

Example

{button ,AL(`OPAL_TYPE_STRING;OPAL_METH_STCHR;OPAL_METH_STCHRTOKEYNAME;',0,"Defaultoverview",)} Related Topics

ansiCode example

The following example assumes that a form contains four field objects: *showAllChars*, *ANSIField*, *OEMField*, and *KeyNameField*. The **keyPhysical** method for *showAllChars* translates each character in the string to its ANSI code, OEM code, and key-name equivalent. These character codes are then written to *ANSIField*, *OEMField*, and *KeyNameField*.

```
; showAllChars::keyPhysical
method keyPhysical(var eventInfo KeyEvent)
var
  anyChar   String
  anyANSI   SmallInt
  anyKeyN   String
  anyOEM    SmallInt
endVar
anyChar = eventInfo.char()           ; get the character typed
anyANSI = ansiCode(anyChar)          ; convert to ANSI code
ANSIField = anyANSI                  ; write ANSI code to ANSIField


anyCode = eventInfo.vCharCode()      ; get the VK_Code of character


anyKeyN = VKCodeToKeyName(anyCode)   ; convert VK_Code to key name
KeyNameField = anyKeyN               ; write key name to KeyNameField


anyOEM = oemCode(anyChar)            ; convert char to OEM code
OEMField = anyOEM                    ; write OEM code to OEMField
beep()
endMethod
```

breakApart method

Splits a string into an array of substrings.

Syntax

```
breakApart ( var tokenArray Array[ ] String [ , const separators String ] )
```

Description

breakApart splits a string into an array of substrings and each substring is written to an element of an array named *tokenArray*. You can specify one or more delimiting characters in *separators*. If you omit *separators*, substrings are delimited by a space. Delimiting characters are not included in *tokenArray*. **breakApart** is especially useful for importing data from a text file into a table.

Note

- Two empty delimiters parse as a token and result in an empty array element.

Example

{button ,AL(`OPAL_TYPE_STRING;OPAL_METH_STSUB;',0,"Defaultoverview",)} Related Topics

breakApart example

In the following example, the **pushButton** method for a button named *breakToArray* creates three arrays from the same string. The first time, the call to the **breakApart** method does not specify delimiters. By default, the method treats spaces as delimiters. The second call to **breakApart** specifies the asterisk as a delimiter. Empty array elements are created each time an asterisk immediately follows another asterisk. The third call specifies question mark, comma, and semicolon as delimiters.

```
; breakToArray::pushButton
method pushButton(var eventInfo Event)
var
  ar Array[] String ; Must be resizable
  s String
endvar


s = "this is, a : delimited ? string"


s.breakApart(ar) ; breaks on spaces by default
ar.view()
{
ar = this
     is,
     a
     :
     delimited
     ?
     string
}


s = "this*is*a*delimited**string"
s.breakApart(ar, "*") ; breaks on specified characters
ar.view()
{
ar = this
     is
     a
     delimited


     string
}


s = "this is, a : delimited ? string"
s.breakApart(ar, ",:?") ; breaks on specified characters
                        ; this time, no space in list of delimiters
ar.view()
{
ar = this is
      a
      delimited
      string
}


endMethod
```

chr procedure

Returns the one-character string represented by an ANSI code.

Syntax

```
chr ( const ansiCode SmallInt ) String
```

Description

chr returns a one-character string containing the ANSI character that corresponds to *ansiCode*. If *ansiCode* is not an integer between 1 and 255, **chr** fails.

You can use **chr** to generate characters that are not easily accessible with the keyboard.

Example

{button ,AL(`OPAL_TYPE_STRING;OPAL_METH_STCHROEM;OPAL_METH_STCHRTOKEYNAME;OPAL_METH_STSTR;',0,"Defaultoverview",)} Related Topics

chr example

In the following example, the **pushButton** method for a button named *showChar* assigns the ANSI character 167 to the *sectionChar* variable. The code then converts character 167 to its key name, assigns it to *sectionKeyName*, and displays both versions of the character in a dialog box.

```
; showChar::pushButton
method pushButton(var eventInfo Event)
var
  sectionChar    String
  sectionKeyName String
endVar
sectionChar = chr(167)                            ; get the character
sectionKeyName = chrToKeyName(chr(167))           ; get the key name
msgInfo("The section character", sectionChar +    ; show the character and
        " has a key name of " + sectionKeyName)   ; the key name
endMethod
```

chrOEM procedure

Returns the one-character string of an OEM code.

Syntax

```
chrOEM ( const oemCode SmallInt ) String
```

Description

chrOEM returns a one-character string containing the OEM character that corresponds to *oemCode*. If *oemCode* is not an integer between 1 and 255, **chrOEM** fails.

You can use **chrOEM** to generate characters that are not easily accessible with the keyboard.

Example

{button ,AL(`OPAL_TYPE_STRING;OPAL_METH_STCHR;OPAL_METH_STSTR;',0,"Defaultoverview",)}

Related Topics

chrOEM example

In the following example, a form has a button named *showOEM* and a field named *fieldOne*. The **pushButton** method for *showOEM* displays the OEM character specified by the number in *fieldOne*.

```
; showOEM::pushButton
method pushButton(var eventInfo Event)
msgInfo("OEM char described by fieldOne", chrOEM(fieldOne))
endMethod
```

chrToKeyName procedure

Returns the virtual key code string of a one-character string.

Syntax

```
chrToKeyName ( const char String ) String
```

Description

chrToKeyName returns the virtual key code of *char* as a string. A key name is a virtual key code (e.g., VK_BACK for Backspace). This method returns the Keyboard constant name as a string (e.g., VK_BACK). Alphanumeric characters and symbols have one-character key names (e.g., J for the letter J).

Example

{button ,AL(`OPAL_TYPE_STRING;OPAL_METH_STVKCODETOKEYNAME;',0,"Defaultoverview",)} Related Topics

chrToKeyName example

See the **chr** example.

fill procedure

Returns a string containing repeated instances of a character.

Syntax

```
fill ( const fillCharacter String, const fillNumber LongInt ) String
```

Description

fill returns a string containing repeated instances of the first character in *fillCharacter* (usually a one-character string), where *fillCharacter* is repeated the number of times specified in *fillNumber*. *fillNumber* must be a non-negative integer. If *fillNumber* is 0, **fill** returns an empty string.

In Corel Paradox 8, the *fillNumber* parameter was changed to a LongInt.

Example

{button ,AL(`OPAL_TYPE_STRING;OPAL_METH_STSPACE;',0,"Defaultoverview",)} **Related Topics**

fill example

In the following example, the **pushButton** method for the *fillAndView* button creates two strings using the **fill** procedure. The first string is created by filling a variable with the same letter five times. The second string is created by repeating the string Shakespeare four times.

```
; fillAndView::pushButton
method pushButton(var eventInfo Event)
var
  str String
endVar
str = fill("X", 5)
str.view()                      ; displays the string XXXXX
str = fill("Shakespeare ", 4)   ; add a space after
                                ; every occurrence
str.view()
; displays: Shakespeare Shakespeare Shakespeare Shakespeare
endMethod
```

format procedure

Controls the format of displayed or printed values.

Syntax

```
format ( const FormatSpec String, const value AnyType ) String
```

Description

format controls the format of displayed or printed values. *formatSpec* is a string expression containing one or more format specifications to be applied to String.

The following table lists the default format specifications and valid data types for each format category. You can also use AnyType values as data types, if the values can be interpreted consistently with the format category.

Format	Meaning	Valid data types	Default
Width	Set allowable field width and decimal precision	All	Entire data value
Alignment	Alignment within width	All	AR (right-aligned) for all numeric types, AL (left-aligned) for all others (including point)
Case	Uppercase or lowercase strings	All string types	No default
Edit	Specify characters and spacing	All numeric types	See following defaults
	Include a specified symbol		No default
	Decimal point character		ED. (period as decimal point)
	Whole number separator		No separator
	Number of leading zeros		None
	Symbol spacing		None
	Scientific notation		No
	Hide trailing spaces		No (show spaces)
	Use zeros as fill pattern		No
	Scale numbers up		No
	Precede with dollar sign		No
	U.S. or Int'l separators	U.S.	
Sign	Format of positive and negative numbers	All numeric	See following
	Positive		No leading positive sign 999
	Negative		Leading minus sign -999
Date	Specify date formats	Date & DateTime	mm/dd/yy(yy) for Date or hh:mm:ss am(pm), mm/dd/yy(yy) for DateTime
Time	Specify time formats	Time & DateTime	hh:mm:ss am(pm) for Date or hh:mm:ss am(pm), mm/dd/yy(yy) for DateTime
Logical	Logical value representation	Logical	True/False

You can combine two or more format specifications in *formatSpec* by separating them with commas.

Type	Spec	Meaning
Width	W*n*	Specifies the total format width, including special characters, leading symbols or spaces, decimal point, and whole number separators
	W.*n*	Specifies the number of decimal places (W12.2 specifies a 12 character field, two of which are after the decimal point)
	W.W	Use decimal places from Windows numbers
	W.$	Use decimal places from Windows currency
Alignment	AL	Left align in field

	AR	Right align in field
	AC	Center in field
Case	CU	Convert to uppercase
	CL	Convert to lowercase
	CC	Convert to initial capitals
Edit	E(*s*)	*s* specifies the symbol that precedes a number
	E$W	Include currency symbol from Windows
	ED*d*	*d* specifies a decimal point character
	EDW	Use the Windows decimal point character
	EN*c*	*c* specifies whole-number separator
	ENW	Use the Windows whole number separator
	EL*n*	*n* specifies the number of leading zeros
	ELW	Use the Windows leading zero setting
	EP0	No symbol spacing
	EP-	Make symbol spacing for negatives
	EP+	Make symbol spacing for positives
	EPB	Make symbol spacing for all numbers
	EPW	Use the Windows symbol spacing setting
	ES	Use scientific notation
	ET	Hide trailing spaces
	EZ	Use zeros as fill pattern
	EB	Use blanks as fill pattern
	E*	Use '*' as fill pattern
	E+*n*	Scale the number up
	E-*n*	Scale the number down
	E$	The same as E($)
	EC	The same as EN (or EN.D)
	EI	The same as ED (or ED,N. if EC is set)
Sign	S+0	Format positives as $999
	S+1	Format positives as +$999
	S+2	Format positives as $+999
	S+3	Format positives as $999+
	S+4	Format positives as 999$
	S+5	Format positives as +999$
	S+6	Format positives as 999+$
	S+7	Format positives as 999$+
	S+8	Format positives as $999DB
	S+W	Format positives as Windows currency
	S-0	Format negatives as ($999)
	S-1	Format negatives as -$999
	S-2	Format negatives as $-999
	S-3	Format negatives as $999-
	S-4	Format negatives as (999$)
	S-5	Format negatives as -999$
	S-6	Format negatives as 999-$
	S-7	Format negatives as 999$-
	S-8	Format negatives as $999CR
	S-W	Format negatives as Windows currency
	SP	The same as S-0

	S-	The same as S-1
	S+	The same as S-1+1
	SC	The same as S-8
	SD	The same as S-8+8
Date	DW1	Day of week as Mon
	DW2	Day of week as Monday
	DWL	Day of week from Windows Long Date
	DM1	Month as 1
	DM2	Month as 01
	DM3	Month as Jan
	DM4	Month as January
	DML	Month from Windows Long Date
	DMS	Month from Windows Short Date
	DD1	Day as 1
	DD2	Day as 01
	DDL	Day from Windows Long Date
	DDS	Day from Windows Short Date
	DY1	Year as 1
	DY2	Year as 01
	DY3	Year as 1901
	DYL	Year from Windows Long Date
	DYS	Year from Windows Short Date
	DO(s)	s specifies order and separators, use %W for weekday,%D for numeric day, %M for month, and %Y for year. Separators are literal (12/28/92 as DO(%W %M-%D-%Y) is Mon 12-28-92)
	DOL	Order and separators as Windows Long Date
	DOS	Order and separators as Windows ShortDate
	D1	Default date format
	D2	As DM4Y3O(%M %D,%Y)
	D3	As DO(%M/%D)
	D4	As DO(%M/%Y)
	D5	As DM3O(%D-%M-%Y)
	D6	As DM3O(%M %Y)
	D7	As DM3Y3O(%D-%M-%Y)
	D8	As DY3O(%M/%D/%Y)
	D9	As DO(%D.%M.%Y)
	D10	As DO(%D/%M/%Y)
	D11	As DO(%Y-%M-%D)
	DEYEA(s)	s specifies A.D. dates
	DEYEB(s)	s specifies B.C. dates
Time	TH1	Hours as 1T
	TH2	Hours as 01
	THW	Hours from Windows
	TM1	Minutes as 1
	TM2	Minutes as 01
	TMW	Minutes from Windows
	TS1	Seconds as 1
	TS2	Seconds as 01
	TSW	Seconds from Windows
	TNA(s)	s is a string that follows times before noon

	TNP(s)	s is a string that follows times after noon
	TNW	Noon settings from Windows
	TO(s)	s specifies the order and separators, use %H for hours, %M for minutes, %S for seconds, %N for am/pm
	TOW	Order and separators from Windows
Logical	LT(s)	s specifies the representation of the logical True value
	LF(s)	s specifies the representation of the logical False value
	LY	Logical values as Yes and No
	LO	Logical values as On and Off

Example

{button ,AL(`OPAL_TYPE_STRING;OPAL_METH_STSTR;',0,"Defaultoverview",)} **Related Topics**

format example

In the following examples assume that a form contains a field named *formatField* and a button named *demoFormat*. The **pushButton** method for *demoFormat* demonstrates different format specifications. In each example, the method fills the *formatField* with the formatted string and displays a copy of the format specification in a dialog box (using view). The method does not move to the next example until the View dialog box is closed, allowing you to examine both the format specification and the formatted output before proceeding.

```
; demoFormat::pushButton
method pushButton(var eventInfo Event)
var
  x  AnyType
  fs, formatField String
endVar


fs = "\"w6\",\"This is a test\""
formatField = format("w6","This is a test")
; displays This i
formatfield.view("format: "+fs)


fs = "\"w7\",1234567"
formatField = format("w7",1234567)
; displays 1234567
formatfield.view("format: "+fs)


fs = "\"w9.2\",1234.567"
formatField = format("w9.2",1234.567)
; displays   1234.57
formatfield.view("format: "+fs)


; Here are some examples of alignment specifications:
fs = "\"w20,ac\",\"This is\""
formatField = format("w20,ac","This is")
; displays              This is
formatfield.view("format: "+fs)

fs = "\"w20,ac\",\"The Title\""
formatField = format("w20,ac","The Title")
; displays            The Title
formatfield.view("format: "+fs)


fs = "\"w20,ac\",\"Of the Book\""
formatField = format("w20,ac","Of the Book")
; displays          Of the Book
formatfield.view("format: "+fs)

fs = "\"w20,al\",123456"
formatField = format("w20,al",123456)
; displays 123456
formatfield.view("format: "+fs)

fs = "\"w20,ar\",123456"
formatField = format("w20,ar",123456)
; displays               123456
formatfield.view("format: "+fs)


; Here are some examples of case specifications:
fs = "\"cu\",\"the quick brown fox\""
formatField = format("cu","the quick brown fox")
; displays THE QUICK BROWN FOX
formatfield.view("format: "+fs)
```

```
fs = "\"cl\",\"JUMPS OVER THE LAZY\""
formatField = format("cl","JUMPS OVER THE LAZY")
; displays jumps over the lazy
formatfield.view("format: "+fs)


fs = "\"cc\",\"dOG.\""
formatField = format("cc","dOG.")
; displays Dog.
formatfield.view("format: "+fs)


fs = "\"cc\",\"widgets'r us \" + \"too\""
formatField = format("cc","widgets'r us " + "too")
; displays Widgets'R Us Too
formatfield.view("format: "+fs)


; Here are some examples of edit specifications:
x = 34567.89
fs = "\"w10.2, e$c\", x"
formatField = format("w10.2, e$c", x)      ; displays $34,567.89
formatfield.view("format: "+fs)


fs = "\"w10.2, e$ci\", x"
formatField = format("w10.2, e$ci", x)     ; displays $34.567,89
formatfield.view("format: "+fs)


fs = "\"w13.2, e$c\", x"
formatField = format("w13.2, e$c", x)      ; displays    $34,567.89
formatfield.view("format: "+fs)


fs = "\"w14.2, e$cb, al\", x"
formatField = format("w14.2, e$cb, al", x) ; displays $   34,567.89
formatfield.view("format: "+fs)


fs = "\"w15.2, e$cz, al\", x"
formatField = format("w15.2, e$cz, al", x) ; displays $0000034,567.89
formatfield.view("format: "+fs)


fs = "\"w15.2, e$c*, al\", x"
formatField = format("w15.2, e$c*, al", x) ; displays $*****34,567.89
formatfield.view("format: "+fs)


; Here are some examples of sign specifications:
x = -3456.12
fs = "\"w8.2, s+\", x"
formatField = format("w8.2, s+", x)          ; displays -3456.12
formatfield.view("format: "+fs)


fs = "\"w11.2, e$c, sc\", x"
formatField = format("w11.2, e$c, sc", x)    ; displays $3,456.12CR
formatfield.view("format: "+fs)


fs = "\"w14.2, e$c*, sp\", x"
formatField = format("w14.2, e$c*, sp", x)   ; displays ($***3,456.12)
formatfield.view("format: "+fs)


fs = "\"w13.2, e$c*, s+\", x"
formatField = format("w13.2, e$c*, s+", x)   ; displays -$***3,456.12
formatfield.view("format: "+fs)


fs = "\"w14.2, e$c*, sd\", x"
formatField = format("w14.2, e$c*, sd", x)   ; displays $***3,456.12CR
formatfield.view("format: "+fs)
```

```
; Here are some miscellaneous examples:
fs = "\"D2\", Date(\"3/7/1948\""
formatField = format("D2", Date("3/7/1948"))   ; displays March 07,1948
formatfield.view("format: "+fs)


fs = "\"W9.2, AL\", 1234.123"
formatField = format("W9.2, AL", 1234.123)
; displays 1234.12
formatfield.view("format: "+fs)


fs = "\"W9.2, AR\", 1234.123"
formatField = format("W9.2, AR", 1234.123)
; displays 1234.12 right aligned in same field
formatfield.view("format: "+fs)


; to display date and time in 24-hour format
fs = "\"TNA(), TNP(), TO(%H:%M:%S %D), DO(%W %M/%D/%Y)\"," +
     " DateTime(\"2:30:00 pm 11/24/92\")"
formatField = format("TNA(), TNP(), TO(%H:%M:%S %D), DO(%W %M/%D/%Y)",
                     DateTime("2:30:00 pm 11/24/92"))
; displays   14:30:00 Tue 11/24/92
formatfield.view("format: "+fs)


; To display a date including the era (B.C. or A.D.):
fs = "\"DEYEA(A.D.)EB(B.C.)O(%M/%D/%Y %E)\",
       date(\"11/15/81\")"
formatField = format("DEYEA(A.D.)EB(B.C.)O(%M/%D/%Y %E)",
                     date("11/15/81"))
; displays 11/15/81 A.D.
formatfield.view("format: "+fs)

endMethod
```

isEmpty

Performs the same function as isBlank.

{button ,AL(`OPAL_TYPE_STRING;OPAL_METH_ATISBL;',0,"Defaultoverview",)} Related Topics

ignoreCaseInStringCompares procedure

Specifies whether to consider case when comparing strings.

Syntax

```
ignoreCaseInStringCompares ( const yesNo Logical )
```

Description

ignoreCaseInStringCompares specifies whether to consider case when comparing strings. By default, string comparisons are case-sensitive (e.g., Q and q are not the same). If you use **ignoreCaseInStringCompares(Yes)**, string comparisons become case0insenesitive. Once you call **ignoreCaseInStringCompares(Yes)**, it stays in effect until you call **ignoreCaseInStringCompares(No)**.

To determine whether case is being considered, use **isIgnoreCaseInStringCompares**.

Example

{button ,AL(`OPAL_TYPE_STRING;OPAL_METH_STISIC;',0,"Defaultoverview",)} **Related Topics**

ignoreCaseInStringCompares example

In the following example, the **pushButton** method for the *tryCompare* button determines whether Corel Paradox is set to ignore case in string comparisons. If **isIgnoreCaseInStringCompares** returns Yes, this code uses **ignoreCaseInStringCompares** to set it to No. The code then compares an uppercase and lowercase string. A message window informs the user that the strings are not equivalent. The code then sets **isIgnoreCaseInStringCompares** to Yes and compares the two strings again, which returns True.

```
; tryCompare::pushButton
method pushButton(var eventInfo Event)
var
  s1,
  s2  String
endVar
s1 = "cat"
s2 = "CAT"
if isIgnoreCaseInStringCompares() then
  ignoreCaseInStringCompares(No)
endIf
x = (s1 = s2)              ; the first  "=" assigns, all others compare
msgInfo(s1 + " = " + s2 + "?", x)    ; displays False
ignoreCaseInStringCompares(Yes)
x = (s1 = s2)
msgInfo(s1 + " = " + s2 + "?", x)    ; displays True
endMethod
```

isIgnoreCaseInStringCompares procedure

Reports whether case is considered when comparing strings.

Syntax

```
isIgnoreCaseInStringCompares ( ) Logical
```

Description

isIgnoreCaseInStringCompares returns True if case is considered when comparing strings; otherwise, it returns False.

To specify whether to consider case, use **ignoreCaseInStringCompares**.

Example

{button ,AL(`OPAL_TYPE_STRING;OPAL_METH_STIGN;',0,"Defaultoverview",)} Related Topics

isIgnoreCaseInStringCompares example

See the **ignoreCaseInStringCompares** example.

isSpace method

Reports whether a string contains white space or is empty.

Syntax

```
isSpace ( const string String ) Logical
```

Description

isSpace returns True if *string* contains only white space or is empty (""); otherwise, it returns False. White space characters include spaces, tabs, carriage returns, linefeeds, and formfeeds.

Example

{button ,AL(`OPAL_TYPE_STRING;OPAL_METH_STSPACE;',0,"Defaultoverview",)} Related Topics

isSpace example

The following example creates several strings and determines whether they contain only white space or are empty. The following code is for the **pushButton** method for the *valString* button:

```
; valString::pushButton
method pushButton(var eventInfo Event)
var
  s String
endVar
s = space(3)                           ; 3 spaces
msgInfo("3 Spaces", s.isSpace())       ; True
s = ""                                 ; empty String
msgInfo("Empty String", s.isSpace())   ; True
s = "Z" + space(2)                     ; Z and 2 spaces
msgInfo("Z and 2 Spaces", s.isSpace()) ; False
endMethod
```

keyNameToChr procedure

Returns the one-character string represented by a virtual key-code string.

Syntax

```
keyNameToChr ( const keyName String ) String
```

Description

keyNameToChr returns the one-character string represented by the virtual key code *keyName*.

keyName must be a Keyboard constant (e.g., VK_BACK for Backspace) but must be supplied as a string (e.g., VK_BACK). Alphanumeric characters and symbols have one-character key names (e.g., J for the letter J).

Example

{button ,AL(`OPAL_TYPE_STRING;OPAL_METH_STCHRTOKEYNAME;OPAL_METH_STKEYNAMETOVKCODE;', 0,"Defaultoverview",)} Related Topics

keyNameToChr example

See the **keyNameToVKCode** example.

keyNameToVKCode procedure

Returns the VK_Code of a virtual key-code string.

Syntax

```
keyNameToVKCode ( const keyName String ) SmallInt
```

Description

keyNameToVKCode returns the virtual key code (VK_Code) of the character represented by the virtual key code *keyName*, given as a string.

keyName must be a Keyboard constant (e.g., VK_BACK for Backspace) but must be supplied as a string (e.g., VK_BACK). Alphanumeric characters and symbols have one-character key names (e.g., J for the letter J).

Example

{button ,AL(`OPAL_TYPE_STRING;OPAL_METH_STCHRTOKEYNAME;OPAL_METH_STKEYNAMETOCHR;OPAL_METH_STVKCODETOKEYNAME;',0,"Defaultoverview",)} Related Topics

keyNameToVKCode example

In the following example, the **pushButton** method for *showCode* sets a string variable named *keyStr* to an open bracket ([). The code then displays the ANSI code and the key name of *keyStr* in a dialog box.

```
; showCode::pushButton
method pushButton(var eventInfo Event)
var
  keyStr  String
endVar
keyStr = "["              ; set the key name for open bracket
msgInfo("VK_Code/Char", "VK_Code: " +           ; VK_Code 91
        String(keyNameToVKCode(keyStr)) +
        "\nCharacter: " + keyNameToChr(keyStr)) ; char "["
endMethod
```

lower method

Converts a string to lowercase letters.

Syntax

lower ( ) String

Description

lower converts a string to lowercase letters. Use **upper** to convert a string to uppercase letters.

Example

{button ,AL(`OPAL_TYPE_STRING;OPAL_METH_STUPP;',0,"Defaultoverview",)} **Related Topics**

lower example

In the following example, the **pushButton** method for *makeLower* creates an uppercase string. The code then uses **lower** to display it in lowercase.

```
; makeLower::pushButton
method pushButton(var eventInfo Event)
var
  myText String
endVar
myText = "HEY, EVERYBODY! IT'S QUITTIN' TIME"
msgInfo("Official Notice", myText.lower())
; displays "hey everybody! it's quittin' time"
endMethod
```

lTrim method

Removes leading blanks from a string.

Syntax

```
lTrim ( ) String
```

Description

lTrim removes spaces and Tab characters from the left end of a string.

Example

{button ,AL(`OPAL_TYPE_STRING;OPAL_METH_STRTRIM;',0,"Defaultoverview",)} **Related Topics**

lTrim example

In the following example, the **pushButton** method for *trimLeft* creates a string with leading spaces and a leading tab (the escape sequence \t). The method displays the original string, uses **lTrim** to remove the leading non-printing characters and then displays the trimmed version.

```
; trimLeft::pushButton
method pushButton(var eventInfo Event)
var
  trimMe, trimmed String
endVar
trimMe = "  \t   First word"  ; string with spaces and a tab
msgInfo("Original string", trimMe)


trimmed = trimMe.lTrim()      ; trim off spaces and tab
msgInfo("A slightly shorter version", trimmed)
; displays "First word"
endMethod
```

match method

Compares a string with a pattern.

Syntax

```
match ( const pattern String [ , var matchVar String ] * ) Logical
```

Description

match compares a string with a pattern. If the string matches the pattern, **match** extracts the components that match the wildcard elements. The value of *pattern* consists of characters interlaced with the wildcard operators .. and @. The .. matches multiple characters (or no characters), and @ matches any single character. **match** ignores or considers case depending on your system settings. Use **isIgnoreCaseInStringCompares** to determine the system setting and use **ignoreCaseInStringCompares** to turn case-sensitivity on or off.

matchVar is a variable to which the matching components are assigned. **match** assigns matched patterns to *matchVar* variables as the patterns are found. The portions of the string matching the wildcard elements are assigned to the variables from left to right. The first matching substring is assigned to the first variable, the second matching substring to the second variable, and so on. If no match is found, variables are not assigned values.

Note

- Quotes in *pattern* require special handling, periods do not. To embed a quote, precede it with a backslash (\"). **match** treats periods as alphanumeric characters.
- Earlier versions of PAL required backslashes to delimit periods.

Example

{button ,AL(`OPAL_TYPE_STRING;OPAL_METH_STAMAT;OPAL_METH_STSEARCH;OPAL_METH_SSIGNORECASEINLOCATE;OPAL_METH_SSISIGNORECASEINLOCATE;',0,"Defaultoverview",)} Related Topics

match example

The following example demonstrates match functionality:

```
var
  s, x, y, z String
endVar


s = "this and that"


msgInfo("match?", s.match("t.."))         ; displays True
msgInfo("match?", s.match("@his.."))      ; displays True
msgInfo("match?", s.match("@ and that"))  ; displays False
msgInfo("match?", s.match("..and.."))     ; displays True


msgInfo("match?", s.match("..and..", x, y))
                     ; displays True (x = this, y = that)


msgInfo("match?", s.match("T..", z))
  ; If isIgnoreCaseInString() is False, this statement displays
  ; False, and z is not assigned. Use
  ; ignoreCaseInStringCompares(Yes) to get this to display
  ; True, and set z to "his and that"
```

oemCode procedure

Returns the OEM code of a one-character string.

Syntax

```
oemCode ( const char String ) SmallInt
```

Description

oemCode returns the OEM code of *char*. *char* is a one-character string. The OEM code is an integer between 1 and 255.

Example

{button ,AL(`OPAL_TYPE_STRING;OPAL_METH_STANSICODE;OPAL_METH_STCHRTOKEYNAME;',0,"Default overview",)} Related Topics

oemCode example

See the **ansiCode** example.

readFromClipboard method

Reads text from the Clipboard.

Syntax

readFromClipboard ( ) Logical

Description

readFromClipboard reads text from the Clipboard. This method reads text in CF_TEXT format.
readFromClipboard returns True if successful; otherwise it returns False.

Example

{button ,AL(`OPAL_TYPE_STRING;OPAL_METH_STWRITETOCLIPBOARD;',0,"Defaultoverview",)} **Related Topics**

readFromClipboard example

In the following example, a form has two buttons: *readFromClipboard* and *writeToClipboard*. The first button reads text from the Clipboard into a String variable that is stored in a table. The second button reads a String value from a table and writes it to the Clipboard.

The following code is attached to the **pushButton** method for *btnReadFromClipboard*:

```
; btnReadFromClipboard::pushButton
method pushButton(var eventInfo Event)
var
   vrString  String
   tcString  TCursor
endVar


   ;// Open table to hold Strings
  tcString.open("mystrings.db")
   if vrString.readFromClipboard() then
      ;// Add a record to the table and insert the value
      tcString.insertRecord()
      tcString.stringField = vrString
      tcString.unlockRecord()
   endIf
   tcString.close()
endMethod
```

The following code is attached to the **pushButton** method for *btnWriteToClipboard*:

```
; btnWriteToClipboard::pushButton
method pushButton(var eventInfo Event)
var
   vrString  String
   tcString  TCursor
endVar


   ;// Open table to which contains strings
   tcString.open("mystrings.db")
   ;// Make sure there is data in the table
   if tcString.nRecords() <> 0 then
      ;// Copy a value to the String variable
      vrString = tcString.stringField
      ;// Write it out to the Clipboard
      vrString.writeToClipboard()
   endIf
   tcString.close()
endMethod
```

rTrim method

Removes trailing blanks from a string.

Syntax

```
rTrim ( ) String
```

Description

rTrim removes spaces, tabs, carriage returns, and linefeed characters from the right end of a string.

Example

{button ,AL(`OPAL_TYPE_STRING;OPAL_METH_STLTRIM;',0,"Defaultoverview",)} Related Topics

rTrim example

In the following example, the **pushButton** method for *trimRight* creates a string with trailing spaces. The code displays the original string, uses **rTrim** to remove the trailing non-printing characters and displays the trimmed version.

```
; trimRight::pushButton
method pushButton(var eventInfo Event)
var
  trimMe, trimmed String
endVar
trimMe = "Last word     "     ; string with trailing spaces
msgInfo("Original string", trimMe + "The end")
; displays "Last word     The end"


trimmed = trimMe.rTrim()      ; trim off spaces
msgInfo("A slightly shorter version", trimmed + "The end")
; displays "Last wordThe end"
endMethod
```

search method

Returns the position of one string inside another string.

Syntax

```
search ( const str String ) SmallInt
```

Description

search searches for *str* within a target string. If *str* is found, **search** returns the starting character position of *str* within the target string; otherwise, it returns 0. The search always begins at the first character of the target string.

By default, **search** is case-sensitive. Use **ignoreCaseInStringCompares** to make the search case-insensitive.

Example

{button ,AL(`OPAL_TYPE_STRING;OPAL_METH_STAMAT;OPAL_METH_STMAT;',0,"Defaultoverview",)}

Related Topics

search example

The following example searches for parts of the string Goliath and Golgolithic. This code is attached to the **pushButton** method for the *searchStr* button.

```
; searchStr::pushButton
method pushButton(var eventInfo Event)
var
  s String
endVar
s = "Goliath"
msgInfo("Where is lia in Goliath?", s.search("lia")) ; displays 3
msgInfo("Where is lai in Goliath?", s.search("lai")) ; displays 0
ignoreCaseInStringCompares(No)
s = "Golgolithic"
msgInfo("Where is gol in Golgolithic?", s.search("gol"))
; displays 4
; Note: If ignoreCaseInStringCompares is on, the last
; search yields a 1 instead.
endMethod
```

searchEx method

Returns the position of one string inside another string.

Syntax

```
searchEx ( const str String ) LongInt
```

Description

searchEx searches for *str* within a target string. Use **searchEx** when working with very large string values. The **searchEx** returns a LongInt, while **search** returns a SmallInt value.

If *str* is found, **searchEx** returns the starting character position of *str* within the target string; otherwise, it returns 0. The search always begins at the first character of the target string.

By default, **searchEx** is case-sensitive. Use ignoreCaseInStringCompares to make it case-insensitive.

Example

{button ,AL(`OPAL_TYPE_STRING;OPAL_METH_STAMAT;OPAL_METH_STMAT;OPAL_METH_STSIZEEX;',0,"Defaultoverview",)} Related Topics

searchEx example

See the search example.

size method

Returns the number of characters in a string.

Syntax

```
size ( ) SmallInt
```

Description

size returns the number of characters (including spaces) in a string as a SmallInt.

Note

- The maximum size of a string has been increased in version 8, and is now limited by available virtual memory only. **size** has been retained for compatibility with existing applications; however, sizeEx (which returns a LongInt) is preferred because it returns the length of both small and large strings.

Example

{button ,AL(`OPAL_TYPE_STRING;OPAL_METH_STSTR;',0,"Defaultoverview",)} Related Topics

size example

In the following example, the **pushButton** method for *getSize* assigns a string to the variable *sourceText*. The code then displays the sentence and its size in a dialog box. The example then uses **size** to retrieve the first half of *sourceText*, and assign it back to *sourceText*. The size of the *sourceText* and the smaller *sourceText* are displayed in a dialog box.

```
; getSize::pushButton
method pushButton(var eventInfo Event)
var
  sourceText String
endVar
sourceText = "This is a short sentence."
msgInfo("Size", "Length: " + String(sourceText.size()) +
                "\n" + sourceText)
; displays   Length: 25
;            This is a short sentence.


; now chop the sentence in half
sourceText = subStr(sourceText, 1, SmallInt(sourceText.size()/2))
msgInfo("Half-Size", "Length: " + strVal(sourceText.size())
                     + "\n" + sourceText)
; displays   Length: 12
;            This is a sh
endMethod
```

sizeEx method

Returns the number of characters in a string.

Syntax

```
sizeEx ( ) LongInt
```

Description

sizeEx returns the number of characters (including spaces) in a string.

Use **sizeEx** when working with very large string values since the returned length is expressed as a LongInt (size returns a SmallInt).

Example

{button ,AL(`OPAL_TYPE_STRING;OPAL_METH_STSTR;OPAL_METH_STLENG;OPAL_METH_STSTRINGEX;',0, "Defaultoverview",)} Related Topics

sizeEx example

See the size example.

space method

Creates a string containing a specified number of spaces.

Syntax

```
space ( const numberOfSpaces LongInt ) String
```

Description

space creates a string containing the number of spaces specified by *numberOfSpaces*.

The *numberOfSpaces* parameter has been changed to LongInt in version 8.

Example

{button ,AL(`OPAL_TYPE_STRING;OPAL_METH_STISPAC;',0,"Defaultoverview",)} **Related Topics**

space example

See the isSpace example.

string procedure

Casts a value as a string.

Syntax

```
string ( const value AnyType [ , const value AnyType ] * ) String
```

Description

string casts a value as a string. If you specify multiple arguments, **string** will cast them all to strings and concatenate them to one string.

Example

{button ,AL(`OPAL_TYPE_STRING;OPAL_METH_STFOR;',0,"Defaultoverview",)} **Related Topics**

string example

In the following example, the **pushButton** method for *getNumToString* requests a number from the user. The code then casts it as a string and concatenates it with another string for display in a **msgInfo** dialog box.

```
; getNumToString::pushButton
method pushButton(var eventInfo Event)
var
   nn Number
endVar
nn = 0.0                     ; initialize the number
nn.View("Enter a number")    ; display it, and ask for input


; Note: Because you can enter only one argument for the text of
; the msgInfo dialog box, if you have any non-string elements, they
; must be cast as strings, then concatenated. Here, nn is cast
; to a String type before being concatenated with "You entered "
msgInfo("Status", "You entered " + string(nn))
msgInfo("Status", string("You entered ", nn))  ; also works
endMethod
```

strVal procedure

Converts a value to a string.

Syntax

```
strVal ( const value AnyType ) String
```

Description

strVal converts *value* to a string. The data type specified in *value* can be an AnyType type.

Example

{button ,AL(`OPAL_TYPE_STRING;OPAL_METH_STSTR;OPAL_METH_STLENG;',0,"Defaultoverview",)}

Related Topics

strVal example

See the **size** example.

subStr method

Returns a portion of a string.

Syntax

```
substr ( const startIndex LongInt [ , const numberOfChars LongInt ] ) String
```

Description

substr returns a portion of a string that starts at *startIndex* and continues for the number of characters specified by *numberOfChars*. The value of *startIndex* must be greater than 0 and less than or equal to the size of the string. If *numberOfChars* is 0, **substr** returns a null string. If *numberOfChars* is omitted, **substr** returns the character that lies at the position specified by *startIndex*.

The *startIndex* and *numberOfChars* parameters have been changed to LongInt in version 8.

Example

{button ,AL(`OPAL_TYPE_STRING;OPAL_METH_STBREAK;OPAL_METH_STSEARCH;OPAL_METH_STLENG;',0,"Defaultoverview",)} Related Topics

subStr example

The following example assumes that a form contains a button named *getPhone* and four fields named *wholePhone*, *phAreaCode*, *phExchange*, and *phNumber*. This example uses **substr** to extract three groups of digits from a U.S. phone number. The following code is attached to the **pushButton** method for *getPhone*:

```
; getPhone::pushButton
method pushButton(var eventInfo Event)
var
  phoneNum  String
endVar
phoneNum = wholePhone.Value
; assume phone number has been entered as ###-###-####
; start from first position, take three characters
phAreaCode.Value = phoneNum.substr(1, 3)  ; get the area code
phExchange.Value = phoneNum.substr(5, 3)  ; get the exchange
phNumber.Value   = phoneNum.substr(9, 4)  ; get the number
beep()
endMethod
```

toANSI method

Converts a string of OEM characters to ANSI characters.

Syntax

```
toANSI ( ) String
```

Description

toANSI converts a string of OEM characters to ANSI characters.

Example

{button ,AL(`OPAL_TYPE_STRING;OPAL_METH_STTOOEM;',0,"Defaultoverview",)} Related Topics

toANSI example

In the following example, the **pushButton** method for a button named *showANSI* displays a string in two ways: as text, in the title of the dialog box and as ANSI code in the window of the dialog box. The last character in the string is the copyright symbol (©). This symbol appears in the title of the dialog box but is replaced by an underscore (_) in the window of the dialog box.

```
; showANSI::pushButton
method pushButton(var eventInfo Event)
var
  ss String
endVar
; string plus copyright symbol
ss = "A string of characters " + chr(169)
msgInfo(ss, ss.toANSI())
; displays string plus "_" in window of dialog box - system-dependent
endMethod
```

toOEM method

Converts a string of ANSI characters to OEM characters.

Syntax

toOEM ( ) String

Description

toOEM converts a string of ANSI characters to OEM characters.

Example

{button ,AL(`OPAL_TYPE_STRING;OPAL_METH_STTOANSI;',0,"Defaultoverview",)} Related Topics

toOEM example

In the following example, the **pushButton** method for a button named *showOEM* displays a string in two ways: as text, in the title of the dialog box and as OEM code in the window of the dialog box. The last character in the string is the copyright symbol (©). This symbol appears in the title of the dialog box but is replaced by an underscore (_) in the window of the dialog box.

```
; showOEM::pushButton
method pushButton(var eventInfo Event)
var
  ss String
endVar
; string plus copyright symbol
ss = "A string of characters " + chr(169)
msgInfo(ss, ss.toOEM())
; displays string plus "c" in window of dialog box
endMethod
```

upper method

Converts a string to uppercase letters.

Syntax

upper ( ) String

Description

upper converts a string to uppercase letters. Use **lower** to convert a string to lowercase letters.

Example

{button ,AL(`OPAL_TYPE_STRING;OPAL_METH_STLOWR;',0,"Defaultoverview",)} Related Topics

upper example

In the following example, the **pushButton** method for *makeUpper* retrieves a string from the user and converts it to uppercase letters. The converted string is then compared to an uppercase string constant.

```
;makeUpper:pushButton
method pushButton(var eventInfo Event)
const
  ORDERTYPE = "BIDORDER"   ; concatenate two valid types
endConst
var
  myText String
  x      SmallInt
endVar
myText = ""                            ; initialize the string
myText.view("Enter 'Bid' or 'Order'") ; get a response
myText = myText.upper()                ; convert to uppercase
if search(ORDERTYPE, myText) > 0 then
  ; search for a matching string -- returns location
  ; of match, or zero if no match
  msgInfo("Status", "You entered a valid type.")
else
  msgStop("Stop", "You must enter either Bid or Order.")
endIf
endMethod
```

vkCodeToKeyName procedure

Converts a virtual key code constant to a virtual key code string.

Syntax

```
vkCodeToKeyName ( const vkCode SmallInt ) String
```

Description

vkCodeToKeyName returns the virtual key code name, as a string, of the character represented by the integer value *vkCode*.

This method returns the name of a Keyboard constant (e.g., VK_BACK for Backspace) as a string (e.g., VK_BACK). Alphanumeric characters and symbols have one-character key names (e.g., J for the letter J).

Example

{button ,AL(`OPAL_TYPE_STRING;OPAL_METH_STANSICODE;OPAL_METH_STCHR;OPAL_METH_STCHRTOKEYNAME;OPAL_METH_STKEYNAMETOCHR;',0,"Defaultoverview",)} Related Topics

vkCodeToKeyName example

See the **ansiCode** example.

writeToClipboard method

Writes a string to the Clipboard.

Syntax

writeToClipboard ( ) Logical

Description

writeToClipboard writes a string to the Clipboard. This method copies strings in the CF_TEXT format.
writeToClipboard returns True if successful and False if unsuccessful. The text copied to the Clipboard is ANSI.

Example

{button ,AL(`OPAL_TYPE_STRING;OPAL_METH_STREADFROMCLIPBOARD;;',0,"Defaultoverview",)}

Related Topics

writeToClipboard example

In the following example, a form has two buttons: *readFromClipboard* and *writeToClipboard*. The first button will read text from the Clipboard into a String variable which will then be stored in a table. The second button read a String value from a table and writes it out to the Clipboard.

The following code is attached to the **pushButton** method for *btnReadFromClipboard*:

```
; btnReadFromClipboard::pushButton
method pushButton(var eventInfo Event)
var
   vrString  String
   tcString  TCursor
endVar


   ;// Open table to hold Strings
  tcString.open("mystrings.db")
   if vrString.readFromClipboard() then
      ;// Add a record to the table and insert the value
      tcString.insertRecord()
      tcString.stringField = vrString
      tcString.unlockRecord()
   endIf
   tcString.close()
endMethod
```

The following code is attached to the **pushButton** method for *btnWriteToClipboard*:

```
; btnWriteToClipboard::pushButton
method pushButton(var eventInfo Event)
var
   vrString  String
   tcString  TCursor
endVar


   ;// Open table to which contains strings
   tcString.open("mystrings.db")
   ;// Make sure there is data in the table
   if tcString.nRecords() <> 0 then
      ;// Copy a value to the String variable
      vrString = tcString.stringField
      ;// Write it out to the Clipboard
      vrString.writeToClipboard()
   endIf
   tcString.close()
endMethod
```

System type

The System type contains methods and procedures for displaying messages, locating system information, setting printer options, manipulating the File Browser, working with the online Help system, and more.

Methods and procedures for the System type

beep
close
compileInformation
constantNameToValue
constantValueToName
cpuClockTime
debug
deleteRegistryKey
desktopMenu
dlgAdd
dlgCopy
dlgCreate
dlgDelete
dlgEmpty
dlgNetDrivers
dlgNetLocks
dlgNetRefresh
dlgNetRetry
dlgNetSetLocks
dlgNetSystem
dlgNetUserName
dlgNetWho
dlgRename
dlgRestructure
dlgSort
dlgSubtract
dlgTableInfo
enableExtendedCharacters
enumDesktopWindowHandles
enumDesktopWindowNames
enumEnvironmentStrings
enumExperts
enumFonts
enumFormats
enumFormNames
enumPrinters
enumRegistryKeys
enumRegistryValueNames
enumReportNames
enumRTLClassNames
enumRTLConstants
enumRTLErrors
enumRTLMethods
enumWindowHandles

enumWindowNames
errorClear
errorCode
errorHasErrorCode
errorHasNativeErrorCode
errorLog
errorMessage
errorNativeCode
errorPop
errorShow
errorTrapOnWarnings
execute
executeString
exit
fail
fileBrowser
fileBrowserEx
formatAdd
formatDelete
formatExist
formatGetSpec
formatSetCurrencyDefault
formatSetDateDefault
formatSetDateTimeDefault
formatSetLogicalDefault
formatSetLongIntDefault
formatSetNumberDefault
formatSetSmallIntDefault
formatSetTimeDefault
formatStringToDate
formatStringToDateTime
formatStringToNumber
formatStringToTime
getDefaultPrinterStyleSheet
getDefaultScreenStyleSheet
getDesktopPreference
getLanguageDriver
getMouseScreenPosition
getRegistryValue
getUserLevel
helpOnHelp
helpQuit
helpSetIndex
helpShowContext
helpShowIndex
helpShowTopic
helpShowTopicInKeywordTable
isErrorTrapOnWarnings
isMousePersistent

message
msgAbortRetryIgnore
msgInfo
msgQuestion
msgRetryCancel
msgStop
msgYesNoCancel
pixelsToTwips
play
projectViewerClose
projectViewerIsOpen
projectViewerOpen
printerGetInfo
printerGetOptions
printerSetCurrent
printerSetOptions
readEnvironmentString
readProfileString
resourceInfo
runExpert
searchRegistry
sendKeys
sendKeysActionID
setDefaultPrinterStyleSheet
setDefaultScreenStyleSheet
setDesktopPreference
setMouseScreenPosition
setMouseShape
setMouseShapeFromFile
setRegistryValue
setUserLevel
sleep
sound
sysInfo
tracerClear
tracerHide
tracerOff
tracerOn
tracerSave
tracerShow
tracerToTop
tracerWrite
twipsToPixels
version
winGetMessageID
winPostMessage
winSendMessage
writeEnvironmentString
writeProfileString

Print related ObjectPAL methods and examples

beep procedure

Sounds the Windows default beep.

Syntax

```
beep ( )
```

Description

beep sounds the Windows default beep. The beep is audible only if a sound device is installed and active.

To play a sound with a specific pitch and duration, use **sound**.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYSOUND;',0,"Defaultoverview",)} **Related Topics**

beep example

The following example prompts you to enter a number and beeps if the number is out of range. This code is attached to a button's **pushButton** method:

```
; getANumber::pushButton
method pushButton(var eventInfo Event)
var
  someNumber  SmallInt
endVar
someNumber = 1
someNumber.view("Pick a number between 1 and 10")
while someNumber < 1 OR someNumber > 10
  beep()          ; beep
  sleep(100)      ; slight pause, otherwise beeps run together as one
  beep()
  msgStop("Oops", "That number is too large or too small. Try again.")
  someNumber.view("Pick a number between 1 and 10")
endwhile
endMethod
```

close procedure

Closes the active form.

Syntax

```
close ( [ const returnValue AnyType ] )
```

Description

close returns a value to the calling form when *returnValue* (optional) is specified. This method does not generate an error if *returnValue* is specified and there is no calling form. Starts the process of closing the form, which includes removing the focus and departing.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYEXIT;',0,"Defaultoverview",)} Related Topics

close example

The following example closes the active form after asking for confirmation:

```
; closeButton::pushButton
method pushButton(var eventInfo Event)
var
  qAnswer String
endVar
qAnswer = msgYesNoCancel("Closing Application",
          "Do you want to close this form?")
if qAnswer = "Yes" then
  close()                 ; close the current form
else
  message("Application not closed.")
endIf
endMethod
```

compileInformation procedure

Lists information about the most recently compiled form.

Syntax

```
compileInformation ( var info DynArray[ ] AnyType )
```

Description

compileInformation lists information about the most recently compiled form. It writes the data to a dynamic array (DynArray) named *info* that you declare and pass as an argument. You can use **compileInformation** for analyzing large forms, libraries, scripts, and reports.

The following table displays the structure of the *info* DynArray:

Index	Definition
CodeSize	Compiled size of the code segment (in bytes).
CompileTime	Compile time (in milliseconds)
DataSize	Compiled size of the data segment (in bytes)
MethodCount	Number of methods that have code and/or comments
SourceSize	Size of the uncompiled source code (in bytes)
SymbolTableSize	Compiled size of the symbol table (in bytes)

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_FOENUMSOURCE;OPAL_METH_FOENUMSOURCETOFILE;OPAL_METH_SYSYSINFO;',0,"Defaultoverview",)} Related Topics

compileInformation example

The following example writes compiler information to a dynamic array *dynCompileInfo*, and then displays it in a **view** dialog box:

```
;analyzeObject::pushButton
method pushButton(var eventInfo Event)
  var
    dynCompileInfo   Dynarray[]   AnyType
  endVar


  compileInformation(dynCompileInfo)
  dynCompileInfo.view()
endmethod
```

constantNameToValue procedure

Returns the numeric value of a constant named *constantName*.

Syntax

```
constantNameToValue ( const constantName String ) AnyType
```

Description

constantNameToValue returns values for predefined ObjectPAL constants only. This method does not return values for user-defined constants.

Note

- For readability, ease of maintenance, and portability, use constants rather than numeric values.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYCONSTANTVALUETONAME;OPAL_METH_SYENUMRTLCONSTANTS;OPAL_ACTIONEVENT_USERDEFINEDCONSTANTS;OPAL_ERROREVENT_USERDEFINEDCONSTANTS;OPAL_MENUEVENT_USERDEFINEDCONSTANTS;',0,"Defaultoverview",)} Related Topics

constantNameToValue example

The following example returns the numeric value for an action constant named DataBeginEdit:

```
; showValOfConst::pushButton
method pushButton(var eventInfo Event)
var
  constValue   AnyType
  constString  String
  tf           Logical
endvar
constValue = constantNameToValue("DataBeginEdit")  ; constant is passed as a
                                                    ; String
msgInfo("The value of DataBeginEdit is", constValue)
tf = constantValueToName("ActionDataCommands", constValue, constString)
if tf then   ; if the conversion worked properly, display the string
  msgInfo("The name of " + String(constValue) + " is", constString)
else
  msgInfo("Status", "Something went wrong with that conversion.")
endIf
endMethod
```

constantValueToName procedure

Reports the name of a constant.

Syntax

```
constantValueToName ( const groupName String, const value AnyType, var constName String )
Logical
```

Description

constantValueToName writes the name of a constant to *constName.* The constant's value equals *value* and that belongs to the group *groupName*, where *groupName* is one of the Types of Constants. This method returns True if successful; otherwise, it returns False.

Works for names of predefined ObjectPAL constants only; not for user-defined constants.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYCONSTANTNAMETOVALUE;OPAL_METH_SYENUMRTLCONSTANTS;OPAL_ACTIONEVENT_USERDEFINEDCONSTANTS;OPAL_ERROREVENT_USERDEFINEDCONSTANTS;OPAL_MENUEVENT_USERDEFINEDCONSTANTS;',0,"Defaultoverview",)} Related Topics

constantValueToName example

See the **constantNameToValue** example.

cpuClockTime procedure

Returns the number of milliseconds that have passed since the computer was booted.

Syntax

cpuClockTime ( ) LongInt

Description

cpuClockTime returns the number of milliseconds that have passed since the computer was booted. The minimum clock increment is 55 milliseconds. This procedure is useful for measuring the interval between two events.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_TITIME;OPAL_TYPE_TIMEREVENT;',0,"Defaultoverview",)}

Related Topics

cpuClockTime example

The following example compares execution times for two **for** loops: one with an undeclared variable, the other with a declared variable. The code executes significantly faster when the variable is declared, although execution times vary by system.

```
; clockVars::pushButton
method pushButton(var eventInfo Event)
var
  fastVar    SmallInt
  delta      String
  startTime,
  stopTime   LongInt
endvar
startTime = cpuClockTime()                ; clock's time before starting
for slowVar from 1 to 10000               ; slowVar is undeclared
  slowVar = slowVar + 1
endFor
stopTime = cpuClockTime()                 ; clock's time after 10000 loops
delta = String(stopTime - startTime)      ; find the elapsed time using
delta.view("Time for undeclared variable") ; an undeclared variable --
                                          ; times vary by system
startTime = cpuClockTime()
for fastVar from 1 to 10000               ; fastVar is declared
  fastVar = fastVar + 1
endFor
stopTime = cpuClockTime()
delta = String(stopTime - startTime)      ; find the elapsed time using
delta.view("Time for declared variable")  ; a declared variable
msgInfo("And the moral is:", "For the best performance, " +
        "declare variables!")
endMethod
```

debug procedure

Halts execution of a method and invokes the Debugger.

Syntax

```
debug ( )
```

Description

debug halts execution of a method and invokes the debugger. **debug** statements have the same effect as setting a breakpoint, although unlike breakpoints, **debug** statements are saved with the method's source code. This procedure is useful for setting persistent breakpoints in methods while you are developing an application.

debug statements are only activated when you click Program, Compile With Debug; otherwise, they are ignored. This allows you to toggle **debug** statements without having to remove them from your code.

Turn Program, Compile With Debug on to test the application.

Turn Program, Compile With Debug off to deliver the application.

Note

- **debug** works only in methods and procedures that you write, not for methods and procedures in the ObjectPAL run-time library.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYEXEC;',0,"Defaultoverview",)} Related Topics

debug example

The following example executes a **for** loop. Halfway through the loop, a call to **debug** suspends execution and opens an Editor window containing the code. Click Program, Run to resume execution, or use the other Debugger features. Assume the command Program, Compile With Debug has been chosen from the ObjectPAL Editor menu.

```
; startDebugAt50::pushButton
method pushButton(var eventInfo Event)
var
  i SmallInt
endVar
for i from 1 to 100
  message(i)
  if i = 50 then
      debug()    ; will work only if Program, Compile With Debug
                 ; ObjectPAL Editor menu command is checked
  endIf
endFor
endMethod
```

deleteRegistryKey method

Deletes a registry key and/or value.

Syntax

```
deleteRegistryKey ( const key String, const value String, const rootKey LongInt ) Logical
```

Description

deleteRegistryKey deletes the registry key specified by *key*. **deleteRegistryKey** returns True if successful; otherwise, it returns False. If the parameter *value* is not empty, *key*'s value name is deleted, but not *key* itself. If *value* is empty, then only *key* is deleted. If *key* has subkeys a warning is generated, and *key* is not deleted.

You can set the *rootKey* with the predefined RegistryKeyType Constants.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYENUMREGISTRYKEYS;OPAL_METH_SYENUMREGISTRYVALUENAMES;OPAL_METH_SYGETREGISTRYVALUE;OPAL_METH_SYSEARCHREGISTRY;OPAL_METH_SYSETREGISTRYVALUE;',0,"Defaultoverview",)} Related Topics

deleteRegistryKey example

The following example adds and then deletes a registry key. If the value parameter is blank, the entire key is deleted; otherwise, the value and corresponding data are deleted.

```
var
   ar Array[] String
endvar


   setRegistryValue( "Software\\Corel\\Paradox\\8.0\\Pdoxwin\\Designer\\MyKey", "MyKeyValue",
"MyKeyData", RegKeyCurrentUser )


   enumRegistryKeys( "Software\\Corel\\Paradox\\8.0\\Pdoxwin\\Designer", RegKeyCurrentUser,
ar )
   ar.view()


   deleteRegistryKey( "Software\\Corel\\Paradox\\8.0\\Pdoxwin\\Designer\\MyKey", "",
RegKeyCurrentUser )


   enumRegistryKeys( "Software\\Corel\\Paradox\\8.0\\Pdoxwin\\Designer", RegKeyCurrentUser,
ar )
   ar.view()
```

desktopMenu procedure

Displays the Corel Paradox desktop menu.

Syntax

```
desktopMenu ( )
```

Description

desktopMenu displays the Corel Paradox desktop menu. This method is useful when you use a form as a dialog box that doesn't have an associated menu.

After you call **desktopMenu**, the Corel Paradox desktop menu persists until:

- the current form or report loses focus
- a call to **removeMenu** restores the default menu for the form or report
- a call to **show** displays a custom menu

Example
{button ,AL(`OPAL_TYPE_SYSTEM;',0,"Defaultoverview",)} Related Topics

desktopMenu example

The following example calls **desktopMenu** in the **setFocus** method on the page of a dialog box to display the Corel Paradox default menu:

```
;pge1 :: setFocus
method setFocus(var eventInfo Event)
   desktopMenu()
endMethod
```

dlgAdd procedure

Displays the Add Records In <table> To dialog box.

Syntax

```
dlgAdd ( const tableName String )
```

Description

dlgAdd displays the Add Records In <table> dialog box.

tableName specifies the source table.

ObjectPAL code suspends execution until the user closes this dialog box.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYDLGCOPY;OPAL_METH_SYDLGEMPTY;OPAL_METH_SYDLGSUBTRACT;',0,"Defaultoverview",)} Related Topics

dlgAdd example

The following example displays the Add Records In <table> To dialog box and inserts the *Customer* table name as the source table. To complete the example, type the target table name and close the dialog box.

```
; showAddDlg::pushButton
method pushButton(var eventInfo Event)
; invoke the Add Records In <table> To dialog box with Customer as the source
dlgAdd("customer.db")
endMethod
```

dlgCopy procedure

Displays the Copy <table> To dialog box.

Syntax

```
dlgCopy ( const tableName String )
```

Description

dlgCopy displays the Copy <table> To dialog box. The argument *tableName* specifies the source table.

ObjectPAL code suspends execution until the user closes this dialog box.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYDLGADD;OPAL_METH_SYDLGEMPTY;OPAL_METH_SYDLGSUBTRACT;',0,"Defaultoverview",)} Related Topics

dlgCopy example

The following example displays the Copy <table> To dialog box and specifies the *Customer* table name as the source table. To complete the example, type the target table name and close the dialog box.

```
; showCopyDlg::pushButton
method pushButton(var eventInfo Event)
; invoke the Copy <table> To dialog box with the Customer table as the source
dlgCopy("customer.db")
endMethod
```

dlgCreate procedure

Displays the Create Table dialog box.

Syntax

```
dlgCreate ( const tableName String )
```

Description

Displays the Create Table dialog box. The argument *tableName* specifies the name of the table to create. When you choose a table type and close the dialog box, this procedure opens a Table Type dialog box for the specified table type.

ObjectPAL code suspends execution until the user closes this dialog box.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYDLGCOPY;OPAL_METH_SYDLGDELETE;',0,"Defaultoverview",)} Related Topics

dlgCreate example

The following example displays the Table Type dialog box. To complete the example, choose the table type, fill out the field roster, and save the table.

```
; showCreateDlg::pushButton
method pushButton(var eventInfo Event)
; invoke the Table Type dialog box -- table name is not used
dlgCreate("sometbl.db")
endMethod
```

dlgDelete procedure

Displays a warning dialog box prompting the user to confirm deletion of the table.

Syntax

```
dlgDelete ( const tableName String )
```

Description

dlgDelete displays a warning dialog box prompting the user to confirm deletion of the table. The argument *tableName* specifies the name of table to delete.

ObjectPAL code suspends execution until the user closes this dialog box.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYDLGCREATE;OPAL_METH_SYDLGEMPTY;',0,"Defaultoverview",)} Related Topics

dlgDelete example

The following example displays a warning dialog box and inserts the *Customer* table name as the table to delete. To complete the example, close the dialog box and confirm the deletion.

```
; showDeleteDlg::pushButton
method pushButton(var eventInfo Event)
; invoke warning dialog box for the Customer table
dlgDelete("Customer.db")  ; same as Tools, Utilities, Delete
endMethod
```

dlgEmpty procedure

Displays a warning dialog box prompting the user to confirm the emptying of the table.

Syntax

```
dlgEmpty ( const tableName String )
```

Description

dlgEmpty displays a warning dialog box prompting the user to confirm the emptying of the table. The argument *tableName* specifies the name of table to empty.

ObjectPAL code suspends execution until the user closes this dialog box.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYDLGDELETE;OPAL_METH_SYDLGSUBTRACT;',0,"Default overview",)} Related Topics

dlgEmpty example

The following example displays the warning dialog box and inserts the *Customer* table name as the table to empty. To complete the example, close the dialog box and confirm the data deletion.

```
method pushButton(var eventInfo Event)
; Displays the warning dialog box for Customer table
dlgEmpty("Customer.db")
endMethod
```

dlgNetDrivers procedure

Opens the Borland Database Engine (BDE) page of the Preferences dialog box.

Syntax

```
dlgNetDrivers ( )
```

Description

dlgNetDrivers opens the BDE page of the Preferences dialog box. ObjectPAL code suspends execution until the user closes the dialog box.

For more information about drivers, see About language drivers.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SSENDRCAP;OPAL_METH_SSENDRINF;OPAL_METH_SSENDRNAM;',0,"Defaultoverview",)} Related Topics

dlgNetDrivers example

The following example opens the BDE page of the Preferences dialog box:

```
; showNetDrivers::pushButton
method pushButton(var eventInfo Event)
; invoke the BDE page of the Preferences dialog box
dlgNetDrivers()
endMethod
```

dlgNetLocks procedure

Creates and displays a table displaying lock information.

Syntax

```
dlgNetLocks ( )
```

Description

dlgNetLocks displays the Select File dialog box and prompts you to choose a table. Click Open to create a Corel Paradox table named LOCKS.DB in your private directory. If the table already exists, Corel Paradox overwrites it without asking for confirmation. If the table is already open, this procedure fails.

Here is the structure of LOCKS.DB:

Field name	Type & size		Description
Type	S	25	Lock type value
Username	A	14	User name of lock owner
Net Session	S		Net level session number
Our Session	S		BDE session number (if the lock is a BDE lock)
Record Number	A	33	Record number of locked record (if Type = Record Lock (Write)

Corel Paradox creates the *Locks* table and displays it in a Table window.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYDLGNETSETLOCKS;OPAL_METH_SYDLGNETRETRY;OPAL_METH_TCENLOC;',0,"Defaultoverview",)} Related Topics

Lock type values for dlgNetLocks (System type)

0 = Record lock
1 = Special record lock
2 = Group lock
3 = Image lock
4 = Table open (no lock)
5 = Table read lock
6 = Table write lock
7 = Table exclusive lock
9 = Unknown lock

dlgNetLocks example

The following example opens the Select File dialog box. After you choose a file, **dlgNetLocks** creates and displays a *Locks* table.

```
; showNetLocks::pushButton
method pushButton(var eventInfo Event)
; creates a table of lock info :PRIV:LOCKS.DB, then displays it
dlgNetLocks()
endMethod
```

dlgNetRefresh procedure

Displays the Database page of the Preferences dialog box.

Syntax

```
dlgNetRefresh ( )
```

Description

dlgNetRefresh displays the Database page of the Preferences dialog box. ObjectPAL code suspends execution until the user closes this dialog box.

For more information, see Database page (Preferences dialog box).

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYDLGNETRETRY;OPAL_METH_SYDLGNETWHO;',0,"Defaultoverview",)} Related Topics

dlgNetRefresh example

The following example opens the Database page of the Preferences dialog box:

```
; showNetRefresh::pushButton
method pushButton(var eventInfo Event)
; invoke the Database page of the Preferences dialog
dlgNetRefresh()
endMethod
```

dlgNetRetry procedure

Displays the Database page of the Preferences dialog box.

Syntax

```
dlgNetRetry ( )
```

Description

dlgNetRetry displays the Database page of the Preferences dialog box. ObjectPAL code suspends execution until the user closes this dialog box.

For more information, see Database page (Preferences dialog box).

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYDLGNETLOCKS;',0,"Defaultoverview",)} Related Topics

dlgNetRetry example

The following example opens the Database page of the Preferences dialog box:

```
; showNetRetryDlg::pushButton
method pushButton(var eventInfo Event)
; invoke the Database page of the Preferences dialog box
dlgNetRetry()
endMethod
```

dlgNetSetLocks procedure

Displays the Table Locks dialog box, allowing you to place a lock on a table.

Syntax

```
dlgNetSetLocks ( )
```

Description

dlgNetSetLocks displays the Table Locks dialog box, allowing you to place a lock on a table. ObjectPAL code suspends execution until the user closes this dialog box.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYDLGNETLOCKS;',0,"Defaultoverview",)} Related Topics

dlgNetSetLocks example

The following example opens the Table Locks dialog box:

```
; showSetLocks::pushButton
method pushButton(var eventInfo Event)
dlgNetSetLocks()  ; invoke the Table Locks dialog box
endMethod
```

dlgNetSystem procedure

Displays the Borland Database Engine (BDE) page of the Preferences dialog box:

Syntax

```
dlgNetSystem ( )
```

Description

dlgNetSystem displays the BDE page of the Preferences dialog box. ObjectPAL code suspends execution until the user closes this dialog box.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYDLGNETDRIVERS;OPAL_METH_SSENDRCAP;OPAL_METH_SSENDRINF;OPAL_METH_SSENDRNAM;',0,"Defaultoverview",)} Related Topics

dlgNetSystem example

The following example opens the Borland Database Engine (BDE) page of the Preferences dialog box:

```
; showNetSystem::pushButton
method pushButton(var eventInfo Event)
; invoke the BDE page of the Preferences dialog box
dlgNetSystem()
endMethod
```

dlgNetUserName procedure

Displays the Database page of the Preferences dialog box. The Database page shows the current user's network name.

Syntax

```
dlgNetUserName ( )
```

Description

dlgNetUserName displays the Database page of the Preferences dialog box. The Database page displays the current user's network name. ObjectPAL code suspends execution until the user closes this dialog box.

For more information, see Database page (Preferences dialog box).

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYDLGNETWHO;',0,"Defaultoverview",)} Related Topics

dlgNetUserName example

The following example opens the Database page of the Preferences dialog box, which shows the current network user's name:

```
; showUserName::pushButton
method pushButton(var eventInfo Event)
; invoke the Database page of the Preferences dialog box
dlgNetUserName()
endMethod
```

dlgNetWho procedure

Displays the Database page of the Preferences dialog box.

Syntax

```
dlgNetWho ( )
```

Description

dlgNetWho displays the Database page of the Preferences dialog box. ObjectPAL code suspends execution until the user closes this dialog box.

For more information, see Database page (Preferences dialog box).

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYDLGNETUSERNAME;',0,"Defaultoverview",)} Related Topics

dlgNetWho example

The following example opens the Database page of the Preferences dialog box:

```
; showUserList::pushButton
method pushButton(var eventInfo Event)
; invoke the Database page of the Preferences dialog box
dlgNetWho()
endMethod
```

dlgRename procedure

Displays the Rename <table> To dialog box.

Syntax

```
dlgRename ( const tableName String )
```

Description

dlgRename displays the Rename <table> To dialog box. The argument *tableName* specifies the table to rename.

ObjectPAL code suspends execution until the user closes this dialog box.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYDLGCOPY;OPAL_METH_SYDLGDELETE;OPAL_METH_SYDLGSORT;',0,"Defaultoverview",)} Related Topics

dlgRename example

The following example displays the Rename <table> To dialog box and specifies *Customer* as the table to rename. To complete the example, type a new name and close the dialog box.

```
; showRenameDlg::pushButton
method pushButton(var eventInfo Event)
; invoke the Table Rename <table> To dialog box
dlgRename("customer.db")
endMethod
```

dlgRestructure procedure

Displays the Restructure Table dialog box.

Syntax

```
dlgRestructure ( const tableName String )
```

Description

dlgRestructure displays the Restructure Table dialog box. The argument *tableName* specifies the table to restructure, including the filename's extension. If *tableName* does not specify a path, **dlgRestructure** searches for the table in the working directory.

If *tableName* does not specify an extension, or specifies an extension of .DB, **dlgRestructure** displays the Restructure Corel Paradox Table dialog box.

If *tableName* specifies an extension of .DBF, **dlgRestructure** displays the Restructure dBASE Table dialog box.

ObjectPAL code suspends execution until the user closes this dialog box.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYDLGCREATE;',0,"Defaultoverview",)} Related Topics

dlgRestructure example

The following example displays the Restructure Table dialog box and specifies *Customer* as the table to restructure. To complete the example, modify the structure and close the dialog box.

```
; showRestructureDlg::pushButton
method pushButton(var eventInfo Event)
; invoke the Restructure Table dialog box for Customer table
dlgRestructure("customer.db")
endMethod
```

dlgSort procedure

Displays the **Sort Table** dialog box.

Syntax

```
dlgSort ( const tableName String )
```

Description

tableName specifies the name of table to sort.

ObjectPAL code suspends execution until the user closes this dialog box.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYDLGRENAME;',0,"Defaultoverview",)} **Related Topics**

dlgSort example

The following example displays the Sort Table dialog box and chooses *Customer* as the table to sort. To complete the example, create a sort specification and close the dialog box.

```
; showSortDlg::pushButton
method pushButton(var eventInfo Event)
; invoke the Sort Table dialog box
dlgSort("customer.db")
endMethod
```

dlgSubtract procedure

Displays the Subtract Records In <table> From dialog box.

Syntax

```
dlgSubtract ( const tableName String )
```

Description

dlgSubtract displays the Subtract Records In <table> From dialog box. The argument *tableName* specifies the table from which to subtract records.

The dialog box opens with the argument *tableName* already specified, prompting the user to choose what to subtract from *tableName*. ObjectPAL code suspends execution until the user closes this dialog box.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYDLGADD;OPAL_METH_SYDLGDELETE;',0,"Defaultoverview",)} Related Topics

dlgSubtract example

The following example displays the Subtract Records In <table> From dialog box and specifies *Customer* as the source table from which to subtract records. To complete the example, close the dialog box.

```
; showSubtractDlg::pushButton
method pushButton(var eventInfo Event)
; invoke the Subtract Records In <table> From dialog box
dlgSubtract("customer.db")  ;
endMethod
```

dlgTableInfo procedure

Displays the Structure Information dialog box.

Syntax

```
dlgTableInfo ( const tableName String )
```

Description

dlgTableInfo displays the Structure Information dialog box. The argument *tableName* specifies the table from which to obtain the structure information.

ObjectPAL code suspends execution until the user closes this dialog box.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYDLGCREATE;OPAL_METH_SYDLGRESTRUCTURE;',0,"Defaultoverview",)} **Related Topics**

dlgTableInfo example

The following example displays the Structure Information dialog box for the *Customer* table:

```
; showTableInfo::pushButton
method pushButton(var eventInfo Event)
; invoke the Structure Information dialog box for the Customer table
dlgTableInfo("customer.db")
endMethod
```

enableExtendedCharacters procedure

Determines whether you can type extended character codes from the numeric keypad without enabling the NumLock key.

Syntax

```
enableExtendedCharacters ( const yesNo Logical ) Logical
```

Description

enableExtendedCharacters determines whether you can type extended character codes from the numeric keypad without enabling the NumLock key. If *yesNo* is set to True, you can type extended characters without NumLock. If *yesNo* is set to False, NumLock must be on to enter extended character codes; otherwise, keypad keys function as navigation keys. This setting affects all forms, and remains active while Corel Paradox is running. This setting is not saved when you exit.

enableExtendedCharacters is used in international applications or other environments where keyboards do not have NumLock keys. This method returns True if successful; otherwise, it returns False.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYSYSINFO;',0,"Defaultoverview",)} **Related Topics**

enableExtendedCharacters example

The following example enables extended characters when the form opens:

```
method open(var eventInfo Event)

   if eventInfo.isPreFilter() then
      ;// This code executes for each object on the form:

   else
      ;// This code executes only for the form:
      doDefault
      enableExtendedCharacters(Yes)
   endIf

endMethod
```

enumDesktopWindowHandles procedure

Lists the window handles of open windows on the Corel Paradox desktop.

Syntax

```
enumDesktopWindowHandles ( var windowHandles DynArray [ ] AnyType [, const className String ] )
```

Description

enumDesktopWindowHandles lists the handles of open windows on the Corel Paradox desktop. This procedure writes the list to a dynamic array (DynArray) named *windowHandles*. The *windowHandles* index contains the handle and specifies the name of the window. The optional *className* argument specifies that the DynArray contains only windows whose *className* equals the name of the window class.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYENUMWINDOWHANDLES;',0,"Defaultoverview",)}

Related Topics

enumDesktopWindowHandles example

The following example builds and displays a dynamic array (DynArray) of all the window titles open on the Corel Paradox desktop:

```
method pushButton(var eventInfo Event)
var
   winHandles DynArray[] String
endvar


enumDesktopWindowHandles(winHandles)   ;// enumerate desktop window
                                       ;// handles to a DynArray
winHandles.view()                      ;// lists all windows open
                                       ;// in the Corel Paradox desktop


endMethod
```

enumDesktopWindowNames procedure

Lists the names of open windows on the Corel Paradox desktop.

Syntax

```
1. enumDesktopWindowNames ( const tableName String ) Logical
2. enumDesktopWindowNames ( const windowNames Array [ ] String [, const className String] )
```

Description

enumDesktopWindowNames lists the names of open windows owned by the Corel Paradox desktop. Syntax 1 creates a Corel Paradox table named *tableName* that lists the name, class, position, and size of each window. If *tableName* does not specify a path, **enumDesktopWindowNames** creates the table in the working directory. If *tableName* already exists, this method overwrites it without asking for confirmation. If *tableName* is open, this method fails.

The following table displays the structure of *tableName*:

Field name	Type & size		Description
WindowName	A	64	Window name (if the window has no name, this field is empty)
ClassName	A	63	Window type
Position	A	12	Coordinates of upper-left corner (e.g.,456, 553)
Size	A	12	Coordinates of lower-right corner (e.g.,889, 221)
Handle	I		Window handle
ChildId	I		ID number of child window (0 = no child window)
ParentHandle	I		Handle of parent window
InstanceHandle	I		Handle of window instance

Syntax 2 fills the array specified by *winArray* with the names of the windows. You must declare *winArray* before calling this method. Applications are listed in Windows z-order: the top window is listed first in the array, the window in the second layer is listed second, and so on. The optional argument *className* specifies that *winArray* displays only the names of windows whose class is equal to *className* .

Compare this method to **enumWindowNames**, which lists all of the Windows applications running on your system.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYENUMFORMNAMES;OPAL_METH_SYENUMREPORTNAMES;OPAL_METH_SYENUMWINDOWNAMES;',0,"Defaultoverview",)} **Related Topics**

enumDesktopWindowNames example

The following example writes the open desktop window titles to an array. The code then creates and displays a table that lists the open desktop window names.

```
; getDesktopWinNames::pushButton
method pushButton(var eventInfo Event)
var
  winNames Array[] String
  tempTV           TableView
endvar
tempTV.open("Customer")               ; open a table view
enumDesktopWindowNames(winNames)      ; enum desktop window names to an array
winNames.view() ; lists all windows open in the Corel Paradox desktop, if
                ; method editor window is open, lists first 32 chars
enumDesktopWindowNames("wNameTbl.db") ; enum to a table
tempTV.open("wNameTbl")               ; show the table
endMethod
```

enumEnvironmentStrings procedure

Lists all of the items from the DOS environment.

Syntax

```
enumEnvironmentStrings ( var values DynArray[ ] String ) Logical
```

Description

enumEnvironmentStrings lists all of the items from the DOS environment. This method writes the items to a dynamic array named *values*, which you declare and pass as an argument.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYRESOURCEINFO;',0,"Defaultoverview",)} Related Topics

enumEnvironmentStrings example

The following example creates and displays a dynamic array named *dyn* that lists items from the DOS environment:

```
;thisButton::pushButton
method pushButton(var eventInfo Event)
  var
    dyn   DynArray[] String
  endVar


  enumEnvironmentStrings(dyn)
  dyn.view()
endmethod
```

enumExperts procedure

Lists all of the experts available to Corel Paradox.

Syntax

```
1. enumExperts ( const expertType String, var expertNames DynArray [ ] AnyType )
2. enumExperts ( const expertType String, const expertName String )
```

Description

enumExperts lists the experts available to Corel Paradox. The *expertType* parameter specifies the type of experts that are included in the list. Syntax 1 fills a dynamic array (DynArray) named *expertNames* with the names of the experts. Syntax 2 lists the experts in a table. The following table displays the format of the table created in Syntax 2:

Field	Type & size		Description
Expert	Alpha	25	Registered expert name
Name	Alpha	25	Visible expert name
Description	Alpha	255	Help description text
File Name	Alpha	255	Expert filename (including the path)
Icon	Graphic		Experts icon graphic

The valid values for *expertType* are:

Document	Identifies document experts (e.g., Table or Form experts)
Object	Identifies experts that are activated when placing an object on a form or report (e.g., the Button Expert)
CoreUI	Identifies experts selected from a menu (e.g., Text Import Expert)

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYRUNEXPERT;',0,"Defaultoverview",)} Related Topics

enumExperts example

The following example enumerates the available experts and determines if expertForm is in the list. If expertForm is available, the code runs the expert.

```
method pushButton(var eventInfo Event)
Var
   da        DynArray[] AnyType
   expertForm    String
endVar


expertForm = "Form"
enumExperts( "Document", da )


if da.contains( expertForm ) then
   runExpert( "Document", expertForm )
else
   msgStop( "Error", "Unable to run the expert:" + expertForm )
endIf
endMethod
```

enumFonts procedure

Creates a table listing the fonts installed on your system.

Syntax

```
1. enumFonts ( const tableName String )
2. enumFonts ( const deviceType SmallInt, var fontList Array[] String )
```

Description

enumFonts creates a table listing the fonts on your system. The argument *tableName* specifies the table. By default, **enumFonts** creates *tableName* in your working directory. If *tableName* already exists, this procedure overwrites it without asking for confirmation. If *tableName* is open, **enumFonts** fails.

The following table displays the structure of *tableName*:

Field name	Type & size		Description
FaceName	A	64	Font name. (e.g., Arial)
FontSize	A	8	Font size in printer's points. (e.g., 12)
Attribute	A	64	Display/print attribute.(e.g., Normal)

Syntax 2 builds an array of fonts in *fontList*. The argument *deviceType* has two possible values: 1 (indicating screen display fonts), and 2 (indicating printer fonts).

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYENUMRTLCONSTANTS;',0,"Defaultoverview",)}

Related Topics

enumFonts example

The following example creates and lists system fonts in a table named FONTS.DB. The code then searches a TCursor for a font named Modern. If Modern is in the table, the code sets the Font.TypeFace property of an unlabeled field object named *balanceField* to Modern.

```
; getFonts::pushButton
method pushButton(var eventInfo Event)
var
  fontsTC TCursor
  tempTV  TableView
endVar
enumFonts("fonts.db")        ; write font names to a table
tempTV.open("fonts.db")      ; show the table
dlgTableInfo("fonts.db")     ; show the table structure
fontsTC.open("fonts.db")
if fontsTC.locate("FaceName", "Modern") then
  balanceField.Font.TypeFace = "Modern"
endIf
fontsTC.close()
endMethod
```

enumFormats procedure

Lists the current formats.

Syntax

```
enumFormats ( const formatType String, var formats DynArray[ ] String ) Logical
```

Description

enumFormats lists the current formats. The data type of the argument *formatType* is Date, Number, Time, DateTime or Logical. This method writes the list to *formats*, a dynamic array that you declare and pass as an argument.

This method returns True if successful; otherwise, it returns False.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_STFOR;',0,"Defaultoverview",)} Related Topics

enumFormats example

The following example creates and displays a dynamic array named *dyn* that lists the formats for Date:

```
; btnInspectFormat :: pushButton
method pushButton(var eventInfo Event)
  var
    s   String
    dyn   DynArray[] String
  endVar


  s = "Date"
  s.view("Enter format to inspect")
  enumFormats(s, dyn)
  dyn.view()
endmethod
```

enumFormNames procedure

Creates an array listing open forms.

Syntax

```
enumFormNames ( var formNames Array[ ] String )
```

Description

enumFormNames creates an array named *formNames* that lists the open forms. You must declare *formNames* as a resizeable array before calling **enumFormNames**. Forms are listed in Windows z-order: the top form is listed first in the array, the form in the second layer is listed second, and so on.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYENUMWINDOWNAMES;OPAL_METH_SYENUMREPORTNAMES;OPAL_METH_SYENUMDESKTOPWINDOWNAMES;',0,"Defaultoverview",)} Related Topics

enumFormNames example

The following example writes the filenames of open forms to an array named *openForms.* The code then displays *openForms*.

```
; getFormNames::pushButton
method pushButton(var eventInfo Event)
var
  openForms Array[] String
endVar
enumFormNames(openForms)
openForms.view()         ; Lists filenames of open forms.
endMethod
```

enumPrinters procedure

Lists the printers installed on your system.

Syntax

```
enumPrinters ( var printers Array[ ] String ) Logical
```

Description

enumPrinters lists the printers installed on your system. **enumPrinters** fills an array named *printers* with elements that each contain the name, driver name, and port (separated by commas) of every printer installed on your system. You must declare *printers* as a resizeable array before calling this method.

If the printer name is Postscript Printer, the driver is PSCRIPT.DRV, and the port is LPT1:

```
PostScript Printer,pscript,LPT1:
```

You pass an array item to **printerSetCurrent** to specify the active printer. Use the String method **breakApart** to separate the components (e.g., to display a list of printer names).

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYPRINTERGETINFO;OPAL_METH_SYPRINTERGETOPTIONS;OPAL_METH_SYPRINTERSETCURRENT;OPAL_METH_SYPRINTERSETOPTIONS;',0,"Defaultoverview",)}

Related Topics

enumPrinters example

The following example retrieves a list of printers installed on your system. If the list includes a PostScript printer, **printerSetCurrent** sets it as the active printer. This example assumes that PostScript printers use a driver named PSCRIPT.DRV.

```
method pushButton(var eventInfo Event)
   var
      arPrinters,
      arPrnNames  Array[] String
      stDrvName,
      stPrnName,
      stPrnInfo   String
      i           SmallInt
   endVar


   stDrvName = "pscript"


   enumPrinters(arPrinters) ; Get a list of installed printers.


   ; See if the list includes a PostScript printer that
  ; uses the "pscript" driver.
   for i from 1 to arPrinters.size()
      stPrnInfo = arPrinters[i]


      ; Info is separated by commas.
      stPrnInfo.breakApart(arPrnNames, ",")


      ; After breakApart, array item 1 is the printer name,
      ; array item 2 is the driver name.
      if arPrnNames[2] = stDrvName then
         ; If a PostScript printer is found, make it current.
         if printerSetCurrent(stPrnInfo) then
            msgInfo("Current printer:", arPrnNames[1])
         else
            errorShow()
         endIf
         return
      endIf
   endFor


   msgStop("Printer setup", "A PostScript printer must be installed.")
endMethod
```

enumRegistryKeys method

Fills an array with keys from the registry.

Syntax

enumRegistryKeys (const ***key*** String, const ***rootKey*** LongInt , var ***keyinfo*** Array[] String, **)** Logical

Description

enumRegistryKeys fills an array with keys from the registry. **enumRegistryKeys** returns True if successful; otherwise, it returns False. An array named *keyinfo* is populated with the full key path from the specified *key* and *rootKey*. The subkeys of *key* are also placed in the array. If *key* is blank, the subkeys of *rootKey* are enumerated.

Set *rootKey* with the predefined RegistryKeyType Constants.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYDELETEREGISTRYKEY;OPAL_METH_SYENUMREGISTRYVALUENAMES;OPAL_METH_SYGETREGISTRYVALUE;OPAL_METH_SYSEARCHREGISTRY;OPAL_METH_SYSETREGISTRYVALUE;',0,"Defaultoverview",)} Related Topics

enumRegistryKeys example

The following example builds an array of the registry keys that contain the string "software\Corel":

```
enumRegistryKeys( "software\\Corel", regKeyLocalMachine, ar )
;\\ values in array
ar[1] = "software\\Corel"
```

The following example displays all registry keys residing under the "Software\Corel\Paradox\8.0\Pdoxwin" key:

```
var
   ar Array[] String
endvar


   enumRegistryKeys( "Software\\Corel\\Paradox\\8.0\\Pdoxwin", RegKeyCurrentUser, ar )
   ar.view()
```

enumRegistryValueNames method

Fills a dynamic array with values and data from the registry.

Syntax

```
enumRegistryValueNames ( const key String, const rootKey LongInt, var keyInfo Array[] String )
Logical
```

Description

enumRegistryValueNames fills an array named *keyInfo* with the value names of the registry specified in *key*. **enumRegistryValueNames** returns True if successful; otherwise, it returns False.

key is entered as a path similar to a file path; however, unlike a file path, wildcards are not expanded. *key* cannot contain a single backslash and cannot be empty. Its maximum size is 65,534 bytes. *keyInfo* contains the value names for the specified *key*. *rootKey* is analogous to a directory drive. Set *rootKey* with the predefined RegistryKeyType Constants.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYDELETEREGISTRYKEY;OPAL_METH_SYENUMREGISTRYKEYS;OPAL_METH_SYGETREGISTRYVALUE;OPAL_METH_SYSEARCHREGISTRY;OPAL_METH_SYSETREGISTRYVALUE;',0,"Defaultoverview",)} Related Topics

enumRegistryValueNames example

The following example lists all the value names under the Software\Corel\Paradox\8.0\Pdoxwin\Designer key. The code assigns the value names and their corresponding values to a DynArray, and displays it:

```
var
   ar      Array[]       String
   dyn   DynArray[]    AnyType
   i      SmallInt
endvar


   enumRegistryValueNames( "Software\\Corel\\Paradox\\8.0\\Pdoxwin\\Designer",
RegKeyCurrentUser, ar )


   if ar.size() > 0 then
      for i from 1 to ar.size()
         dyn[ ar[ i ] ] = getRegistryValue( "Software\\Corel\\Paradox\\8.0\\Pdoxwin\\Designer",
ar[ i ], RegKeyCurrentUser )
      endfor
   endif


   dyn.view()
```

enumReportNames procedure

Creates an array listing open reports.

Syntax

```
enumReportNames ( var reportNames Array[ ] String )
```

Description

enumReportNames fills an array named *reportNames* with the names of open reports in your desktop. You must declare *reportNames* as a resizeable array before calling this method. Reports are listed in Windows z-order — the top report is listed first in the array, the report in the second layer is listed second, and so on.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYENUMFORMNAMES;OPAL_METH_SYENUMDESKTOPWINDOWNAMES;OPAL_METH_SYENUMWINDOWNAMES;',0,"Defaultoverview",)} Related Topics

enumReportNames example

The following example writes the open report names to an array named *openReports* and then displays the array:

```
; getReportNames::pushButton
method pushButton(var eventInfo Event)
var
  openReports Array[] String
endVar
enumReportNames(openReports)
openReports.view()           ; lists open reports
endMethod
```

enumRTLClassNames procedure

Creates a table listing the object types or classes known to ObjectPAL.

Syntax

```
enumRTLClassNames ( const tableName String ) Logical
```

Description

enumRTLClassNames creates a table named *tableName* listing the object types (classes) in the ObjectPAL run-time library. By default, **enumRTLClassNames** saves *tableName* in the working directory. If *tableName* already exists, **enumRTLClassNames** overwrites it without asking for confirmation. If *tableName* is open, **enumRTLClassNames** fails. This method returns True if successful; otherwise, it returns False.

The following table displays the structure of *tableName*:

Field name	Type & size		Description
ClassName	A	32	ObjectPAL type name. (e.g., UIObject)

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYENUMRTLCONSTANTS;OPAL_METH_SYENUMRTLERRORS;OPAL_METH_SYENUMRTLMETHODS;OPAL_TYPE_CATEGORIES;',0,"Defaultoverview",)} Related Topics

enumRTLClassNames example

The following example writes the run-time library class names to a table named *Rtlclass.* The code then displays the table.

```
; getRTLClasses::pushButton
method pushButton(var eventInfo Event)
var
  tempTV TableView
endVar
enumRTLCLassNames("rtlclass.db")      ; write class names to table
tempTV.open("rtlclass")               ; show the table
endMethod
```

enumRTLConstants procedure

Creates a table listing the constants defined by ObjectPAL.

Syntax

```
enumRTLConstants ( const tableName String ) Logical
```

Description

enumRTLConstants creates a table named *tableName* listing all the constants defined in the ObjectPAL run-time library. By default, **enumRTLConstants** creates *tableName* in the working directory. If *tableName* already exists, **enumRTLConstants** overwrites it without asking for confirmation. If *tableName* is open, **enumRTLConstants** fails.

The following table displays the structure of *tableName*:

Field name	Type & size		Description
GroupName*	A	32	One of the types of constants (e.g., ActionDataCommands)
ConstantName*	A	48	Symbolic name of the constant (e.g., DataArriveRecord)
Type	A	48	Data type of the constant (e.g., SmallInt)
Value	A	64	Value of the constant (e.g., 3111)

(* = key field)

Note

- Although Corel Paradox provides the constant's values, refer to constants by name in your code. Use the **constantValueToName** and **constantNameToValue** methods to convert values and constants.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYENUMRTLCLASSNAMES;OPAL_METH_SYENUMRTLERRORS;OPAL_METH_SYENUMRTLMETHODS;',0,"Defaultoverview",)} Related Topics

enumRTLConstants example

The following example writes the run-time library constant descriptions to a table named *Rtlconst*. The code then displays the table.

```
; getRTLConsts::pushButton
method pushButton(var eventInfo Event)
var
  tempTV TableView
endVar
enumRTLConstants("rtlconst.db")     ; write constants names to table
tempTV.open("rtlconst")             ; show the table
endMethod
```

enumRTLErrors procedure

Lists the error codes and messages used by ObjectPAL.

Syntax

```
enumRTLErrors ( const tableName String ) Logical
```

Description

enumRTLErrors creates a table named *tableName* listing the error codes and messages used by ObjectPAL. By default, **enumRTLErrors** creates *tableName* in the working directory. If*tableName* already exists, **enumRTLErrors** overwrites it without asking for confirmation.

The following table displays the structure of *tableName*:

Field	Type & size		Description
ErrorNo*	N		Error number (decimal)
ErrorNoX	A	8	Error number (hex)
Name	A	48	Error constant name, if it exists (e.g., peNoMemory). If an error constant name does not exist, the Name field displays the following string<<Unmapped Error>>
Value	M	230	Error message (e.g., Insufficient memory for this operation)

(* = key field)

This method returns True if successful; otherwise, it returns False. If you pass **enumRTLErrors** an invalid table name, this procedure fails and returns False.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYENUMRTLCLASSNAMES;OPAL_METH_SYENUMRTLCONSTANTS;OPAL_METH_SYENUMRTLMETHODS;',0,"Defaultoverview",)} Related Topics

enumRTLErrors example

The following example writes the run-time library error codes and descriptions to a table named *Rtlerror.* The code then displays the table.

```
;getRTLErrors::pushButton
method pushButton(var eventInfo Event)
  var
    tv  TableView
  endVar


  enumRTLErrors("RTLerror.db")
  tv.open("RTLerror.db")
endMethod
```

enumRTLMethods procedure

Creates a table listing the RTL methods and RTL procedures in ObjectPAL.

Syntax

```
enumRTLMethods ( const tableName String ) Logical
```

Description

enumRTLMethods creates a table named *tableName* listing the RTL methods and procedures used by ObjectPAL. By default, **enumRTLMethods** creates *tableName* in the working directory. If*tableName* already exists, **enumRTLMethods** overwrites it without asking for confirmation. If *tableName* is open, **enumRTLMethods** fails.

The following table displays the structure of *tableName*:

Field name	Type & size	Description
ClassName*	A 32	ObjectPAL type name (e.g., FileSystem)
MethodType*	A 8	Method (for methods) or Proc (for procedures)
MethodName*	A 64	Name of method or procedure (e.g., isDir)
MethodArgs*	A 255	Arguments to the method or procedure (e.g., const dirName String)
ReturnType*	A 32	Data type of returned value or blank if no return value (e.g., Logical)

(* = key field)

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYENUMRTLCLASSNAMES;OPAL_METH_SYENUMRTLCONSTANTS;OPAL_METH_SYENUMRTLERRORS;',0,"Defaultoverview",)} Related Topics

enumRTLMethods example

The following example writes the run-time library method descriptions to a table named *Rtlmeth.* The code then displays the table.

```
; getRTLMethods::pushButton
method pushButton(var eventInfo Event)
var
  tempTV TableView
endVar
enumRTLMethods("rtlmeth.db")   ; write method names to table
tempTV.open("rtlmeth")         ; show the table
endMethod
```

enumWindowHandles procedure

Lists the open window handles.

Syntax

```
enumWindowHandles ( var windowHandles DynArray [ ] AnyType [, const className String ] )
```

Description

enumWindowHandles lists the handles of the open windows running under Windows. This procedure writes the list to a dynamic array (DynArray) named *windowHandles*. The *windowHandles* index contains the handle and the value is the name of the window. The optional *className* argument specifies that the generated list contains only those windows whose *className* equals the name of the window class.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYENUMDESKTOPWINDOWHANDLES;',0,"Defaultoverview",)} Related Topics

enumWindowHandles example

The following example builds and displays a dynamic array(DynArray) of all the window handles:

```
method pushButton(var eventInfo Event)
var
   winHandles DynArray[] String
endvar


enumWindowHandles(winHandles)   ;// enumerate desktop window
                                ;// handles to a DynArray
winHandles.view()               ;// lists all open windows


endMethod
```

enumWindowNames procedure

The following example creates a list of the applications currently running under Windows.

Syntax

```
1. enumWindowNames ( const tableName String ) Logical
2. enumWindowNames ( var windowNames Array [ ] String [, const className String] )
```

Description

enumWindowNames creates a list of applications currently running under Windows. Syntax 1 creates a table named *tableName* listing the name, class, position, size, and handles to each open application on your system. By default, **enumWindowNames** creates *tableName* in the working directory. If*tableName* already exists, **enumWindowNames** overwrites it without asking for confirmation. If *tableName* is open, **enumWindowNames** fails.

The following table displays the structure of *tableName*:

Field name	Type & size		Description
WindowName	A	64	Name of window, or blank if no name
ClassName	A	64	Window type
Position	A	12	Coordinates of upper-left corner (e.g., 456, 553).
Size	A	12	Coordinates of lower-right corner (e.g., 889, 221).
Handle	I		Window handle
ChildId	I		ID number of child window (0 = no child window)
ParentHandle	I		Handle of parent window
InstanceHandle	I		Handle of window instance

Syntax 2 fills an array named *winArray* with the names of all current applications, in Windows z-order. the top application is listed first in the array, the application in the second layer is listed second, and so on. You must declare *winArray* before calling this procedure. An optional argument named *className* specifies that only those windows whose class is equal to *className* appear in *winArray*.

Compare this method to **enumDesktopWindowNames**, which lists only the open windows owned by Corel Paradox.

Examples

enumWindowNames examples

Example1 Using the **pushButton** method to display window information

Example2 Using the **pushButton** method to search for window information

enumWindowNames example 1

The following example uses the **pushButton** method for a button named *getWindowNames* to write and display open window information in two ways. First, it fills an array with the titles of the open windows and displays the array. Next, it fills a table with descriptions of the open windows, and displays the table.

```
; getWindowNames::pushButton
method pushButton(var eventInfo Event)
var
  winNames Array[] String
  tempTV           TableView
endvar
enumWindowNames(winNames)       ; write names to an array
winNames.view()                 ; lists all open windows
                                ; if a method editor window is open,
                                ; lists first 32 chars
enumWindowNames("wNameTbl.db")  ; write window descriptions to a table
tempTV.open("wNameTbl")         ; show the table
dlgTableInfo("wNameTbl.db")     ; show the table structure
endMethod
```

enumWindowNames example 2

The following example uses the **pushButton** method for a button named *btnCalc* to write the open window information to a table named *:PRIV:APPS.DB*. The code then searches the table for *Calculator.* If *Calculator* is found, the code uses the Windows API call **bringToTop** (registered in the Uses window of the button) to switch to *Calculator*. If *Calculator* is not found, the **pushButton** method executes *CALC.EXE*.

```
;btnCalc :: Uses
uses USER32
   BringWindowToTop(WinHandle CWORD)
endUses


;btnCalc :: pushButton
method pushButton(var eventInfo Event)
   var
      stApps       String
      tc           TCursor
      siWinHandle  SmallInt
   endVar


   stApps = ":PRIV:APPS.DB"
   enumWindowNames(stApps)


   tc.open(stApps)


   if tc.locate("WindowName", "Calculator") then
      siWinHandle = tc.handle
      BringWindowToTop(siWinHandle)
   else
      execute("CALC.EXE")
   endIf
endmethod
```

errorClear procedure

Clears the error stack.

Syntax

```
errorClear ( )
```

Description

errorClear clears the error stack of all error codes and error messages.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYERCOD;OPAL_METH_SYERMES;OPAL_METH_SYERRORPOP;',0,"Defaultoverview",)} Related Topics

errorClear example

The following example clears the error stack:

```
; clearError::pushButton
method pushButton(var eventInfo Event)
errorClear()           ; clear the error stack
endMethod
```

errorCode procedure

Returns a number representing the most recent run-time error or error condition.

Syntax

errorCode ( ) SmallInt

Description

errorCode returns a number representing the most recent run-time error or error condition. ObjectPAL provides error constants for these integers (e.g., peObjectNotFound). Use **enumRTLErrors** to create a list of error codes and error messages.

Calling **errorCode** is not the same as calling **eventInfo.setErrorCode**, which adds error information to the event packet, but not to the error stack.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYERRORHASERRORCODE;OPAL_METH_SYERRORHASNATIVEERRORCODE;OPAL_METH_SYERRORLOG;OPAL_METH_SYERMES;OPAL_METH_SYERRORPOP;OPAL_METH_SYENUMRTLERRORS;',0,"Defaultoverview",)} Related Topics

errorCode example

The following example uses a **try** clause to attempt to attach to an object *boxOne* to the current form. If the object doesn't exist, a critical error occurs, and control moves to the **onFail** clause. The **onFail** clause uses **errorCode** to identify the error and then takes appropriate action.

```
; handleErrorcode::pushButton
method pushButton(var eventInfo Event)
var
  obj UIObject
endVar
try
  obj.attach("boxOne")
  obj.color = Red
onFail
  if errorCode() = peObjectNotFound then
     obj.create(BoxTool, 180, 180, 360, 360)
     obj.name = "boxOne"
     obj.visible = Yes
     reTry
  else
     fail()
  endIf
endTry
endMethod
```

errorHasErrorCode method

Searches for a specific error code in the error stack.

Syntax

```
errorHasErrorCode ( const errCode SmallInt ) Logical
```

Description

errorHasErrorCode searches the error stack for the error specified by *errCode. errCode* is an Errors constant or a user-defined error constant. **errorHasErrorCode** returns True if the error is found; otherwise, it returns False.

Use **enumRTLErrors** to create a list of error codes and error messages.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYERCOD;OPAL_METH_SYERRORHASNATIVEERRORCODE; OPAL_METH_SYERRORLOG;OPAL_METH_SYERMES;OPAL_METH_SYERRORPOP;',0,"Defaultoverview",)}

Related Topics

errorHasErrorCode example

The following example searches the error stack for a key violation:

```
if errorHasErrorCode(peKeyViol) then

     ; error handling code goes here

endIf
```

errorHasNativeErrorCode method

Searches for an SQL error code the error stack.

Syntax

```
errorHasNativeErrorCode ( const errCode LongInt ) Logical
```

Description

errorHasNativeErrorCode searches the error stack for an SQL error code. The SQL error is specified by the argument *errCode*. Error codes vary depending on the server and may overlap with some Corel Paradox error codes. **errorHasNativeErrorCode** returns True if the error is found; otherwise, it returns False.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYERRORHASERRORCODE;OPAL_METH_SYENUMRTLERRORS;OPAL_METH_SYERCOD;OPAL_METH_SYERRORLOG;OPAL_METH_SYERMES;OPAL_METH_SYERRORNATIVECODE;OPAL_METH_SYERRORPOP;OPAL_METH_SYENUMRTLERRORS;',0,"Defaultoverview",)} Related Topics

errorHasNativeErrorCode example

The following example searches the error stack for the server error associated with the peServerPathIllegal constant. The constant is set to an error code listed in the server's documentation:

```
if errorHasNativeErrorCode(peServerPathIllegal) then

     ; error handling code goes here

endIf
```

errorLog procedure

The following example adds error information to the error stack.

Syntax

```
errorLog ( const errorCode SmallInt, const errorMessage String )
```

Description

errorLog adds error information to the error stack. Use Errors constants or user-defined error constants to specify the value of *errorCode*. Use **enumRTLErrors** to create a list of error codes and error messages.

Calling **errorLog** is not the same as calling **eventInfo.setErrorCode**, which adds error information to the event packet, but not to the error stack.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYERCOD;OPAL_METH_SYERRORHASERRORCODE;OPAL_METH_SYERRORHASNATIVEERRORCODE;OPAL_METH_SYERMES;OPAL_METH_SYERRORNATIVECODE;OPAL_METH_SYERRORPOP;',0,"Defaultoverview",)} Related Topics

errorLog example

The following example uses a **try** clause to attempt to attach to an object *boxOne* to the current form. If the object doesn't exist, a critical error occurs, and control moves to the **onFail** clause. If the error code isn't peObjectNotFound, the method creates and logs a custom error.

```
; pushMessage::pushButton
method pushButton(var eventInfo Event)
var
  obj    UIObject
  eCode  LongInt
  eMsg   String
endVar
try
  obj.attach("boxOne")
  obj.color = "RedBlue"   ; invalid color constant--will cause an error
                          ; other than peObjectNotFound
onFail
  if errorCode() = peObjectNotFound then
    msgInfo("And the error was", errorMessage())
    obj.create(BoxTool, 180, 180, 360, 360)
    obj.name = "boxOne"
    obj.visible = Yes
    reTry
 else
   ; pop off the original error
   eCode = errorCode()
   eMsg = errorMessage()
   errorPop()
   ; push the original error back onto the stack, but
   ; modify the error message
   errorLog(eCode, self.Name + "::pushButton failed at " +
            String(time()) + ". " + eMsg)
   msgInfo("And the new error is", errorMessage())
   fail()
 endIf
endTry
endMethod
```

errorMessage procedure

Returns a string containing the most recent run-time error message or error condition from the error stack.

Syntax

```
errorMessage ( ) String
```

Description

errorMessage returns a string containing the most recent run-time error message or error condition from the error stack. This method returns the empty string ("") if no error has occurred. **errorMessage** is especially useful for logging error messages during a session.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYERCOD;OPAL_METH_SYERRORHASERRORCODE;OPAL_METH_SYERRORHASNATIVEERRORCODE;OPAL_METH_SYERRORLOG;OPAL_METH_SYERRORNATIVECODE;OPAL_METH_SYERRORPOP;OPAL_METH_SYENUMRTLERRORS;',0,"Defaultoverview",)} Related Topics

errorMessage example

See the **errorLog** example.

errorNativeCode method

Returns the SQL server's error code.

Syntax

errorNativeCode ( ) LongInt

Description

errorNativeCode returns the SQL server's error code. The SQL server's error code varies depending on the server and might overlap some Corel Paradox error codes. If **errorCode** returns the constant peGeneralSQL, **errorNativeCode** returns the server's error code. **errorNativeCode** usually returns zero.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYERRORHASERRORCODE;OPAL_METH_SYERRORHASNATIVEERRORCODE;OPAL_METH_SYERMES;',0,"Defaultoverview",)} Related Topics

errorNativeCode example

The following example determines whether a server has error occurred. If a server error has occurred, the code displays the error code.

```
if errorCode() = peGeneralSQL then
   message("SQL server error number " + string(errorNativeCode()))
endIf
```

errorPop procedure

Removes the most recently added error code and error message from the error stack.

Syntax

```
errorPop ( ) Logical
```

Description

errorPop removes the most recently added error code and error message from the error stack. This procedure allows you to access the stack layer below the current layer.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYERRORHASERRORCODE;OPAL_METH_SYENUMRTLERRORS;OPAL_METH_SYERCOD;OPAL_METH_SYERRORLOG;OPAL_METH_SYERMES;OPAL_METH_SYERRORNATIVECODE;OPAL_METH_SYENUMRTLERRORS;OPAL_METH_SYERRORSHOW;',0,"Defaultoverview",)} Related Topics

errorPop example

See the **errorLog** example.

errorShow procedure

Displays the current error information in the Error log box.

Syntax

errorShow ([const ***topHelp*** String [, const ***bottomHelp*** String]] **)** Logical

Description

errorShow displays the current error information in the Error log box. The argument *topHelp* labels the top portion of the dialog box, and *bottomHelp* the bottom.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYERRORHASERRORCODE;OPAL_METH_SYENUMRTLERRORS;OPAL_METH_SYERCOD;OPAL_METH_SYERRORLOG;OPAL_METH_SYERMES;OPAL_METH_SYERRORNATIVECODE;OPAL_METH_SYERRORPOP;OPAL_METH_SYENUMRTLERRORS;',0,"Defaultoverview",)} Related Topics

errorShow example

The following example uses a button named *tryAnError* to log several errors onto the error stack, and uses **errorShow** to display them:

```
; tryAnError::pushButton
method pushButton(var eventInfo Event)
; add two errors to the error stack
errorLog(1, "First error")
errorLog(2, "Second error")
; show the error dialog box (error 2 shows first)
errorShow("Title for top", "Title for bottom")
endMethod
```

errorTrapOnWarnings procedure

Specifies whether to handle warning errors as critical errors.

Syntax

```
errorTrapOnWarnings ( const yesNo Logical )
```

Description

errorTrapOnWarnings specifies whether to handle warning errors as critical errors. By default, warning errors are not trapped in a **try...onFail** block. If you set the argument *yesNo* to Yes, **errorTrapOnWarnings** traps warning errors as critical errors. This procedure affects only the active form.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYERRORLOG;OPAL_METH_SYERCOD;OPAL_METH_SYERMES;OPAL_METH_SYERRORSHOW;',0,"Defaultoverview",)} Related Topics

errorTrapOnWarnings example

The following example attempts to open an invalid form. If **errorTrapOnWarnings** is set to yes, and error message Is produced; otherwise, no message is produced.

```
; warningToError::pushButton
method pushButton(var eventInfo Event)
var
  someForm Form
endVar
someForm.open("someFile.fsl") ; attempt to attach to a nonexistent form
                              ; normally, this doesn't cause an error
errorTrapOnWarnings(Yes)      ; set the trap
someForm.open("someFile.fsl") ; this time, you get an error message
errorTrapOnWarnings(No)       ; restore to normal
endMethod
```

execute procedure

Executes a program or DOS command.

Syntax

```
execute ( const programName String [ , const wait Logical [ , const displayMode SmallInt ] ] )
Logical
```

Description

execute executes a program or DOS command. The argument *programName* specifies the program or DOS command to be launched. An optional argument named *wait* specifies whether ObjectPAL suspends execution until you close the program. An optional ExecuteOption named *displayMode* specifies the video display mode used when executing the command.

If you have to specify a path to *programName*'s directory, use double backslashes (\\) in the path names.

Note

- The *wait* statement is not valid when executed within the WindowsNT environment.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYPLAY;',0,"Defaultoverview",)} Related Topics

execute example

The following example launches the Windows Clock application in the default window and waits for you to close it before resuming execution:

```
; showClock::pushButton
method pushButton(var eventInfo Event)
   execute("clock.exe", Yes, ExeShowNormal)  ; execute Windows Clock
endMethod
```

executeString method

Converts a string to an ObjectPAL script and runs the script.

Syntax

```
executeString ( const scriptText String [, const otherText String] ) AnyType
```

Description

executeString converts a string to an ObjectPAL script and runs the script. This method inserts the string in the script's built-in **run** method. You can declare types, constants, and variables within the string. The optional *otherText* argument allows you to include ObjectPAL constructs (e.g., procedures or a Uses clause). The *otherText* argument refers to constructs included before the script's built-in **run** method.

To return a value from **executeString**, use **formReturn**.

If the string contains syntax errors, the Script window remains on the desktop.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYEXEC;',0,"Defaultoverview",)} Related Topics

executeString example

The following example calls a routine from Windows and runs it:

```
method run(var eventInfo Event)
var
   msgText, usesText  string
endvar


; Note the backslash char protects quotes inside the quoted string

msgText = "MessageBoxA(0,\"A Message\", \"Hello World\" , 1)"


usesText = "Uses USER32
      MessageBoxA(hwnd CLONG,
                  str1 CPTR,
                  str2 CPTR,
                  boxType CLONG) CLONG
         endUses"
   ;// Now display the message box
executeString(msgText, usesText)
endMethod
```

exit procedure

Exits the Corel Paradox application.

Syntax

```
exit ( )
```

Description

exit closes Corel Paradox. If you try to exit Corel Paradox without saving your changes, **exit** prompts you to save your work.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYCLOSE;',0,"Defaultoverview",)} **Related Topics**

exit example

The following example creates an Exit button which asks for confirmation and closes Corel Paradox:

```
; btnExit::pushButton
method pushButton(var eventInfo Event)
   var
      stQuit  String
   endVar


   stQuit  = msgYesNoCancel("Exit", "Do you want to quit?")
   if stQuit = "Yes" then
      exit() ; If user chooses Yes, then exit.
   endIf
endMethod
```

fail procedure

Causes a method to fail.

Syntax

```
fail ( [ const errorNumber SmallInt, const errorMessage String ] )
```

Description

fail causes a method to fail. Executing **fail** in the **onFail** section of a **try...onFail** block forces a jump to the next highest block (if one exists). **fail** then jumps to the implicit **try...onFail** block that ObjectPAL wraps around every method. Use an Errors constant or a user-defined error constant to set a value for *errorNumber*, which specifies an error code on failure. *errorMessage* (optional) specifies a displayed error message.

enumRTLErrors creates a list of error codes and error messages.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_BLANG_BLTRY;OPAL_METH_SYERCOD;OPAL_METH_SYERMES;OPAL_METH_SYERRORSHOW;',0,"Defaultoverview",)} Related Topics

fail example

See the **errorCode** example.

fileBrowser

See **fileBrowserEx**.

fileBrowserEx replaces **fileBrowser** in this version. **fileBrowser** still functions, and can still be used when working with forms created using older versions of Corel Paradox. However, only **fileBrowserEx** guarantees that the full filename, including its drive or alias, is returned.

fileBrowserEx procedure

Displays the Corel Paradox File Browser and returns the names of the files you select.

Syntax

```
1. fileBrowserEx ( var selectedFile String [ , var browserInfo FileBrowserInfo ] ) Logical
2. fileBrowserEx ( var selectedFiles Array[ ] String [ , var browserInfo FileBrowserInfo ] ) Logical
```

Description

fileBrowserEx suspends ObjectPAL execution until you close the File Browser. This method returns True if you select at least one file; otherwise, it returns False (even if you click OK to close the dialog box).

Use Syntax 1 to return one filename in *selectedFile.* Use Syntax 2 to return an array of filenames in the resizeable array *selectedFiles*.

In either syntax, you can provide an optional record that specifies the data that the File Browser displays. For example, you can instruct the File Browser to display Corel Paradox tables only, forms only, forms and reports, and so on. ObjectPAL provides a special predefined Record structure called FileBrowserInfo that you use only with the **fileBrowserEx** procedure.

The following table displays the structure of **FileBrowserInfo:**

Field	Type	Description
Title	String	The dialog box title
Options	LongInt	Handling instructions for the filename that the user inputs
AllowableTypes	LongInt	The permitted file types, based on file extensions
SelectedType	LongInt	One of the allowable types
FileFilters	String	The file specification in the edit box
CustomFilter	String	One or more file masks in the Files Of Type list box. Each file mask contains the list box text, and the file mask. The two parts are separated by a delimiter character (|). The mask can include any valid filename character, and the ? and * wildcard characters. To display all Corel Paradox tables, use the following custom filter: "Corel Paradox tables|*.db|". To display a list box that allows you to display all Corel Paradox tables or all dBASE tables, use the following custom filter: "Corel Paradox tables|*.db|dBASE tables|*.dbf|". The string's last character determines the delimiter.
Alias	String	The alias or drive name listed in the Alias box
Path	String	The path of the selected file or files. The value is returned by the File Browser and cannot be set directly.
Drive	String	The drive of the selected file or files. The value is returned by the File Browser and cannot be set directly.
DefaultExt	String	The default file extension. Use DefaultExt and NewFileOnly to allow users to omit the file extension when naming a new file.
PathOnly	Logical	The path only of the selected file or files, without filename.
NewFileOnly	Logical	If True, the File Browser behaves like the Save As dialog box. If False, the File Browser behaves like the Open dialog box.

This record structure is built into ObjectPAL. Simply declare a variable of type FileBrowserInfo and assign values to the fields in its structure.

When you call **fileBrowserEx**, values from the File Browser dialog box are inserted in the Alias, Path, and Drive fields. This allows you to determine what you selected in the File Browser.

The AllowableTypes field specifies what appears in the list box for the Types panel in the File Browser. The SelectedType field indicates which of the AllowableTypes is currently selected. Use FileBrowserFileTypes constants for values in the SelectedType and AllowableTypes fields.

Because the **fileBrowserEx** procedure affects the field names in a structure, you can pass to it a simpler record structure that contains only those which interest you.

Examples

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_TYPE_FILESYSTEM;OPAL_CONST_COFILEBROWSERFILETYPES;',0,"Defaultoverview",)} Related Topics

fileBrowserEx procedure examples

Example1 Using **fileBrowserEx** to open a window and return an array of names

Example2 Using **fileBrowserInfo** to pass information

fileBrowserEx example 1

The following example calls **fileBrowserEx** twice. First, **fileBrowserEx** returns one filename. If that filename is a table name, **fileBrowserEx** opens a Table window. Next, **fileBrowserEx** returns an array of filenames and displays the array in a dialog box. The array of filenames is selected by pressing SHIFT and clicking files.

```
; fileBrowserExButton::pushButton
method pushButton(var eventInfo Event)
var
   oneFile           String
   manyFiles Array[] String
   tView             TableView
endVar
fileBrowserEx(oneFile)   ; display the File Browser, and wait
                         ; for you to choose one file
                         ; variable oneFile stores the filename chosen
if isTable(oneFile) then
   tView.open(oneFile) ; open a Table window for the chosen file
endIf


fileBrowserEx(manyFiles) ; let you select multiple files and store
                         ; the filenames in an array
manyFiles.view()         ; displays your choices
endMethod
```

fileBrowserEx example 2

The following example uses a FileBrowserInfo record to pass information. Attach the following code to a button's built-in **pushButton** method. When it executes, this code displays the Browser, waits for you to choose a file, and displays information about your choice in the status area.

```
method pushButton(var eventInfo Event)
var
   fbi FileBrowserInfo ; Declare a variable that uses the predefined
                       ; FileBrowserInfo record structure
   selectedFile String
endVar

; The following statements assign values to fields in the

; record of file browser information

fbi.Alias = ":WORK:" ; Search the current working directory
fbi.AllowableTypes = fbTable + fbForm ; Search for tables and forms
fbi.CustomFilter = "(Bitmap image) *.bmp|*.bmp|(Other graphics files) *.jpg;*.pcx|
*.jpg;*.pcx||"


; Display the Browser and process your selection
if fileBrowserEx(selectedFile, fbi) then
   message("You selected ", selectedFile)
else
   message("You selected cancel")
endIf


endMethod
```

formatAdd procedure

Adds a format.

Syntax

```
formatAdd ( const formatName String, const formatSpec String ) Logical
```

Description

formatAdd adds a format. It creates a format named *formatName* which is described by *formatSpec*. The new format is available to the current session. This method returns True if successful; otherwise, it returns False.

Note

- **formatAdd** does not save Field width (W*n*), Alignment (AR, AL, AC), and Case specifiers (CU, CL, CC) in the new format definition. However, save decimal precision (W.*n*) is preserved. See **format** in the String type for a complete description of format specifiers.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYFORMATDELETE;OPAL_METH_SYFORMATEXIST;',0,"Defaultoverview",)} Related Topics

formatAdd example

The following example adds a new format specification to the session and then sets the default Currency format to the new format:

```
; addAFormat::pushButton
method pushButton(var eventInfo Event)
var
  someNum Currency
endVar
; first, add a currency format with 4 decimal digits and
; a floating dollar sign (windows dollar sign)
formatAdd("FourCurrency", "W.4, E$W")
; then, set the default format for Currency to the new format
formatSetCurrencyDefault("FourCurrency")
someNum = 41324.09876
someNum.view()                  ; appears as $41,324.0988
endMethod
```

formatDelete procedure

Deletes a format.

Syntax

```
formatDelete ( const formatName String ) Logical
```

Description

formatDelete deletes the format specified by the argument *formatName* from the current session.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_STFOR;OPAL_METH_SYFORMATADD;OPAL_METH_SYFORMATEXIST;',0,"Defaultoverview",)} Related Topics

formatDelete example

The following example deletes the custom format named *FourCurrency*:

```
; deleteAFormat::pushButton
method pushButton(var eventInfo Event)
if formatExist("FourCurrency") then
  formatDelete("FourCurrency")
else
  msgInfo("FYI", "Format was not found.")
endIf
endMethod
```

formatExist procedure

Reports whether a format exists.

Syntax

```
formatExist ( const formatName String ) Logical
```

Description

formatExist reports whether the format *formatName* is available in the current session. This method returns True if the format is available; otherwise, it returns False.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_STFOR;OPAL_METH_SYFORMATADD;OPAL_METH_SYFORMATDELETE;',0,"Defaultoverview",)} Related Topics

formatExist example

The following example determines whether a custom format named *FourCurrency* exists. If*FourCurrency* does not exist, the code adds the format specification and displays a number formatted in the new format.

```
; addCurrFormatExist::pushButton
method pushButton(var eventInfo Event)
var
  someNum Currency
endVar
; check if custom format exists already
if NOT formatExist("FourCurrency") then
  ; if not, add a currency format with 4 decimal digits and
  ; a floating dollar sign (windows dollar sign)
  msgInfo("FYI", "Format does not exist. Adding it now.")
  formatAdd("FourCurrency", "W.4, E$W")
else
  msgInfo("FYI", "Format already exists.")
endIf
; set the default format for Currency to the new format
formatSetCurrencyDefault("FourCurrency")
someNum = 41324.09876
someNum.view()        ; displays number as $41324.0988, because
                      ; someNum is a variable of Currency type
endMethod
```

formatGetSpec procedure

Returns the format specification for a named format.

Syntax

```
formatGetSpec ( const formatName String ) String
```

Description

formatGetSpec returns the format specification for the format specified by *formatName.* You can pass the return value to **formatStringToDate** and **formatStringToNumber** to format a string into a date or number.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_STFOR;OPAL_METH_SYFORMATDELETE;OPAL_METH_SYFORMATEXIST;',0,"Defaultoverview",)} Related Topics

formatGetSpec example

The following example uses **formatGetSpec** and **formatStringToDate** to assign a date to a Date type variable the Windows Long format:

```
;Btn :: pushButton
method pushButton(var eventInfo Event)
   var
      d  Date
   endVar


   d = formatStringToDate("Friday, January 08, 1965", formatGetSpec("Windows Long"))
   d.view()
endMethod
```

formatSetCurrencyDefault procedure

Sets the default display format for Currency values.

Syntax

```
formatSetCurrencyDefault ( const formatName String ) Logical
```

Description

formatSetCurrencyDefault sets the default display format for Currency values. This setting remains in effect throughout the session.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_STFOR;OPAL_METH_SYFORMATSETNUMBERDEFAULT;',0,"Defaultoverview",)} Related Topics

formatSetCurrencyDefault example

See the **formatExist** example.

formatSetDateDefault procedure

Sets the default display format for Date values.

Syntax

```
formatSetDateDefault ( const formatName String ) Logical
```

Description

formatSetDateDefault sets the default display format for Date values. This setting remains in effect throughout the session.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_STFOR;OPAL_METH_SYFORMATSETTIMEDEFAULT;OPAL_METH_SYFORMATSETDATETIMEDEFAULT;',0,"Defaultoverview",)} Related Topics

formatSetDateDefault example

The following example uses the **pushButton** method for a button named *setDateFormat* to set the default display format for Date values to the Windows Long format. The code then displays a date in the new format:

```
; setDateFormat::pushButton
method pushButton(var eventInfo Event)
var
  someDate Date
endVar
if formatExist("Windows Long") then
  formatSetDateDefault("Windows Long")
  someDate = date("9/15/92")
  someDate.view()              ; displays "Tuesday, September 15, 1992"
else
  msgStop("Stop", "Requested format does not exist.")
endIf
endMethod
```

formatSetDateTimeDefault procedure

Sets the default display format for DateTime values.

Syntax

```
formatSetDateTimeDefault ( const formatName String ) Logical
```

Description

formatSetDateTimeDefault sets the default display format for DateTime values. This setting remains in effect throughout the session.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_STFOR;OPAL_METH_SYFORMATSETDATEDEFAULT;OPAL_METH_SYFORMATSETTIMEDEFAULT;',0,"Defaultoverview",)} Related Topics

formatSetDateTimeDefault example

The following example uses the **pushButton** method for a button named *setDateTimeFormat* to set the default display format for DateTime values. The code then uses **view** to display a DateTime value in the new format:

```
setDateTimeFormat::pushButton
method pushButton(var eventInfo Event)
var
  someDateTime DateTime
endVar
if formatExist("h:m:s am m/d/y") then
  formatSetDateTimeDefault("h:m:s am m/d/y")
  someDateTime = DateTime("11:45:25 am 11/24/61")
  someDateTime.view()                  ; displays 11:45:25 AM 11/24/61
else
  msgInfo("Status", "Requested format does not exist.")
endIf
endMethod
```

formatSetLogicalDefault procedure

Sets the default display format for Logical values.

Syntax

```
formatSetLogicalDefault ( const formatName String ) Logical
```

Description

formatSetLogicalDefault sets the default display format for Logical values. This setting remains in effect throughout the session.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_STFOR;OPAL_METH_SYFORMATADD;',0,"Defaultoverview",)} Related Topics

formatSetLogicalDefault example

The following example uses the **pushButton** method for a button named *setLogicalFormat* to set the default display format for Logical values to the Male/Female format. The code then displays a logical value in the new format.

```
; setLogicalFormat::pushButton
method pushButton(var eventInfo Event)
var
  someLogical  Logical
endVar
if formatExist("Male/Female") then
  formatSetLogicalDefault("Male/Female")
  someLogical = True
  someLogical.view()               ; displays Male
else
  msgStop("Stop", "Requested format does not exist.")
endIf
endMethod
```

formatSetLongIntDefault procedure

Sets the default display format for LongInt values.

Syntax

```
formatSetLongIntDefault ( const formatName String ) Logical
```

Description

formatSetLongIntDefault sets the default display format for LongInt values. This setting remains in effect throughout the session.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_STFOR;OPAL_METH_SYFORMATSETSMALLINTDEFAULT;',0, "Defaultoverview",)} Related Topics

formatSetLongIntDefault example

The following example uses the **pushButton** method for a button named *setIntegerFormat* to set the default display format for LongInt values to the Integer format. The code then displays a long integer in the new format.

```
; setIntegerFormat::pushButton
method pushButton(var eventInfo Event)
var
  someInt  LongInt
endVar
if formatExist("Integer") then
  formatSetLongIntDefault("Integer")
  someInt = 238756
  someInt.view()                       ; displays 238756
else
  msgStop("Stop", "Requested format does not exist.")
endIf
endMethod
```

formatSetNumberDefault procedure

Sets the default display format for Number values.

Syntax

```
formatSetNumberDefault ( const formatName String ) Logical
```

Description

formatSetNumberDefault sets the default display format for Number values. This setting remains in effect throughout the session.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_STFOR;OPAL_METH_SYFORMATSETCURRENCYDEFAULT;',0,"Defaultoverview",)} Related Topics

formatSetNumberDefault example

The following example uses the **pushButton** method for a button named *setNumberFormat* to set the default display format for Number values to the Scientific format. The code then displays a number in the new default format.

```
; setNumberFormat::pushButton
method pushButton(var eventInfo Event)
var
  someNum Number
endVar
if formatExist("Scientific") then
  formatSetNumberDefault("Scientific")
  someNum = 3489.283
  someNum.view()             ; Displays 3.489283e+3.
else
  msgStop("Stop", "Requested format does not exist.")
endIf
endMethod
```

formatSetSmallIntDefault procedure

Sets the default display format for SmallInt values.

Syntax

```
formatSetSmallIntDefault ( const formatName String ) Logical
```

Description

formatSetSmallIntDefault sets the default display format for SmallInt values. This setting remains in effect throughout the session.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_STFOR;OPAL_METH_SYFORMATSETLONGINTDEFAULT;',0,"Defaultoverview",)} Related Topics

formatSetSmallIntDefault example

The following example uses the **pushButton** method for a button named *setSmallIntFormat* to set the default display format for SmallInt values to the Integer format. The code then displays a small integer in the new default format.

```
; setSmallIntFormat::pushButton
method pushButton(var eventInfo Event)
var
  someInt  SmallInt
endVar
if formatExist("Integer") then
  formatSetSmallIntDefault("Integer")
  someInt = 324
  someInt.view()                    ; displays 324
else
  msgStop("Stop", "Requested format does not exist.")
endIf
endMethod
```

formatSetTimeDefault procedure

Sets the default display format for Time values.

Syntax

```
formatSetTimeDefault ( const formatName String ) Logical
```

Description

formatSetTimeDefault sets the default display format for Time values. This setting remains in effect throughout the session.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_STFOR;OPAL_METH_SYFORMATSETDATEDEFAULT;OPAL_METH_SYFORMATSETDATETIMEDEFAULT;',0,"Defaultoverview",)} Related Topics

formatSetTimeDefault example

The following example uses the **pushButton** method for a button named *setTimeFormat* to set the default display format for Time values to the format *hh:mm:ss am*. The code then display a time in the new default format.

```
; setTimeFormat::pushButton
method pushButton(var eventInfo Event)
var
  someTime Time
  someStr  String
endVar
if formatExist("hh:mm:ss am") then
  formatSetTimeDefault("hh:mm:ss am")
  someTime = time("12:22:45 pm")
  someTime.view()                    ; displays 12:22:45 PM
else
  msgInfo("Status", "Requested format does not exist.")
endIf
endMethod
```

formatStringToDate procedure

Uses a format specification to translate a String value to a Date value.

Syntax

```
formatStringToDate ( dateString String, formatSpec String ) Date
```

Description

formatStringToDate uses a format specification to translate a String value to a Date value. This method translates *dateString* (a string value representing a date) to a Date type value using to the format specification in *formatSpec*. This method returns the Date value and leaves the String value unmodified.

formatSpec is the format specification of a named format, not the format name itself. To retrieve the format specification of a named format, use **formatGetSpec**.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_STFOR;OPAL_METH_SYFORMATDELETE;OPAL_METH_SYFORMATEXIST;OPAL_METH_SYFORMATGETSPEC;OPAL_METH_SYFORMATSTRINGTODATETIME;OPAL_METH_SYFORMATSTRINGTONUMBER;OPAL_METH_SYFORMATSTRINGTOTIME;',0,"Defaultoverview",)} Related Topics

formatStringToDate example

The following example formats a String value as a valid date. The code is attached to the built-in **changeValue** method of an Alpha field and executes when you type a value and leave the field (e.g., press ENTER).

If the field object is bound to a Date field (instead of an Alpha field), Corel Paradox validates the date without writing ObjectPAL code.

```
method changeValue(var eventInfo ValueEvent)
   var
      stUserDate   String
      daValidDate  Date
   endVar

   doDefault

   ; Assume user enters "09-94-23" into this Alpha field object.
   stUserDate = self.Value

   try
      ; Format your value as a valid date.
      daValidDate = formatStringToDate(stUserDate, "DO(%M-%Y-%D)")

      ; formatStringToDate does not change the String value.
      ; It returns a Date value. The following statement displays
      ;         You entered: 09-94-23
      ;         Valid date: 09/23/94

      msgInfo("You entered: " + stUserDate,
            "Valid date: " + String(daValidDate))

   onFail
      ; If user's value cannot be formatted as a date,
      ; display a message.
      msgStop(stUserDate, "Cannot format that value as a Date.")
   endTry

endMethod
```

formatStringToDateTime method

Translates a String value to a DateTime value.

Syntax

```
formatStringToDateTime ( const dateTimeString String, const formatSpec String ) DateTime
```

Description

formatStringToDateTime translates *dateTimeString* to a DateTime value, using the format specification in **formatSpec**. If successful, **formatStringToDateTime** returns a DateTime value and leaves the *dateTimeString* value unmodified. The value of *formatSpec* must be the format specification of a named format, not the format name. To retrieve the format specification of a named format, use **formatGetSpec**.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_STFOR;OPAL_METH_SYFORMATDELETE;OPAL_METH_SYFORMATEXIST;OPAL_METH_SYFORMATGETSPEC;OPAL_METH_SYFORMATSTRINGTODATE;OPAL_METH_SYFORMATSTRINGTONUMBER;OPAL_METH_SYFORMATSTRINGTOTIME;',0,"Defaultoverview",)} Related Topics

formatStringToDateTime example

The following example converts the specified string to the DateTime data type and displays the result:

```
view( formatStringToDateTime( "23:59:59, 3/23/99", "TH1O(%H:%M:%S, %D)" ) )
```

formatStringToNumber procedure

Uses a format specification to translate a String value to a Number value.

Syntax

formatStringToNumber (*numberString* String, *formatSpec* String) Number

Description

formatStringToNumber translates *numberString* (a string value that represents a number) to a Number value, using the format specification in *formatSpec*. If successful, this procedure returns the Number value and leaves the String value unmodified.

The value of *formatSpec* must be the format specification of a named format, not the format name. To retrieve the format specification of a named format, use **formatGetSpec**.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_STFOR;OPAL_METH_SYFORMATDELETE;OPAL_METH_SYFORMATEXIST;OPAL_METH_SYFORMATSTRINGTODATE;',0,"Defaultoverview",)} Related Topics

formatStringToNumber example

In the following example, two strings are concatenated to form a number in scientific notation format. **formatStringToNumber** is used to assign the value to a Number variable and the formatted and unformatted values are displayed in a dialog box. **formatStringToNumber** assigns the formatted value to a Number variable, but leaves the String value unmodified.

```
;btnScientific :: pushButton
method pushButton(var eventInfo Event)
   var
      st1,
      st2,
      stSciNot  String
      nuResult  Number
   endVar


   st1 = "1.e"
   st2 = "+2"
   stSciNot = st1 + st2
   nuResult = formatStringToNumber(stSciNot, "S-4")

   ; The following statement displays
   ; Before format: 1.e+2
   ; After format: 100.00
   msgInfo("Before format: " + stSciNot,
           "After format: " + String(nuResult))
endMethod
```

formatStringToTime method

Translates a String value to a Time value.

Syntax

```
formatStringToTime (const timeString String, const formatSpec String ) Time
```

Description

formatStringToTime translates *timeString* to a Time value, using the format specification in *formatSpec*. If successful, **formatStringToTime** returns a Time value and leaves the String value unmodified. The value of *formatSpec* must be the format specification of a named format, not the format name. To retrieve the format specification of a named format, use **formatGetSpec**.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_STFOR;OPAL_METH_SYFORMATDELETE;OPAL_METH_SYFORMATEXIST;OPAL_METH_SYFORMATGETSPEC;OPAL_METH_SYFORMATSTRINGTODATE;OPAL_METH_SYFORMATSTRINGTODATETIME;OPAL_METH_SYFORMATSTRINGTONUMBER;',0,"Defaultoverview",)} Related Topics

formatStringToTime example

The following example converts the specified string to the Time data type and displays the result:

```
view( formatStringToTime( "23:59:59", "TH1O(%H:%M:%S)" ) )
```

getDefaultPrinterStyleSheet procedure

Returns the name of the default printer style sheet used by documents designed for the printer.

Syntax

getDefaultPrinterStyleSheet ( ) String

Description

getDefaultPrinterStyleSheet returns the name of the default printer style sheet used by documents designed for the printer. If the style sheet is in the working directory, **getDefaultPrinterStyleSheet** returns the filename and extension (e.g., COREL.FP). If the style sheet resides in another directory, **getDefaultPrinterStyleSheet** returns the full path (e.g., C:\COREL\SUITE8\PARADOX\COREL.FP).

Use **getStyleSheet** and **setStyleSheet** for forms and reports that use different style sheets..

Use **getDefaultScreenStyleSheet** to retrieve the default screen style sheet. This screen style sheet is used when you create design documents for the screen.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYSETDEFAULTSCREENSTYLESHEET;OPAL_METH_FOSETSTYLESHEET;OPAL_METH_FOSAVESTYLESHEET;OPAL_METH_FOGETPROTOPROPERTY;OPAL_METH_FOSETPROTOPROPERTY;',0,"Defaultoverview",)} Related Topics

getDefaultPrinterStyleSheet example

See the **setDefaultPrinterStyleSheet** example.

getDefaultScreenStyleSheet procedure

Returns the name of the default screen style sheet used by design documents that are created for the screen.

Syntax

```
getDefaultScreenStyleSheet ( ) String
```

Description

getDefaultScreenStyleSheet returns the filename of the default style sheet for screen documents. If the style sheet is in the working directory, **getDefaultScreenStyleSheet** returns the filename and extension (e.g., COREL.FP). If the style sheet resides in another directory, **getDefaultScreenStyleSheet** returns the full path (e.g., C:\COREL\SUITE8\PARADOX\COREL.FP).

Use **getStyleSheet** and **setStyleSheet** for forms and reports that use different style sheets.

Use **getDefaultPrinterStyleSheet** to retrieve the name of the default printer style sheet, used whenever you create design documents that are designed for the printer.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYSETDEFAULTPRINTERSTYLESHEET;OPAL_METH_SYSETDEFAULTSCREENSTYLESHEET;OPAL_METH_FOSAVESTYLESHEET;OPAL_METH_FOGETPROTOPROPERTY;OPAL_METH_FOSETPROTOPROPERTY;',0,"Defaultoverview",)} Related Topics

getDefaultScreenStyleSheet example

See the **setDefaultScreenStyleSheet** example.

getDesktopPreference procedure

Retrieves a desktop preference value.

Syntax

```
getDesktopPreference (const section AnyType, const name AnyType) AnyType
```

Description

getDesktopPreference returns the value of the desktop preference specified by the *section* and *name* arguments. The *value* returned corresponds to one of the DesktopPreferenceTypes Constants.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYSETDESKTOPPREFERENCE;',0,"Defaultoverview",)}

Related Topics

getDesktopPreference example

The following example displays the sets the title name preference and then retrieves and displays the name:

```
method pushButton(var eventInfo Event)
setDesktopPreference( PrefProjectSection, prefTitleName,"Corel Paradox pour Windows" )

x = getDesktopPreference( PrefProjectSection, prefTitleName )

x.view()
endmethod
```

getLanguageDriver procedure

Returns the default language driver name for the system.

Syntax

```
getLanguageDriver ( ) String
```

Description

getLanguageDriver returns the default language driver name for the system.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_TCGETLANGUAGEDRIVER;OPAL_METH_QUSETLANGUAGEDRIVER;',0,"Defaultoverview",)} Related Topics

getLanguageDriver example

The following example displays the system's language driver name on the Status Bar:

```
;btnDefaultDriver :: pushButton
method pushButton(var eventInfo Event)
  message(getLanguageDriver())
endmethod
```

getMouseScreenPosition procedure

Returns the mouse position as a Point data type.

Syntax

```
getMouseScreenPosition ( ) Point
```

Description

getMouseScreenPosition returns the coordinates (in twips) of the pointer relative to the screen. Use Point type methods (e.g., **x** and **y**) to retrieve more information.

getMouseScreenPosition retrieves the mouse position at the precise time of an event. The coordinates of the current mouse position might be different.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYSMPOS;OPAL_METH_MEGPOS;OPAL_METH_MESPOS;',0, "Defaultoverview",)} Related Topics

getMouseScreenPosition example

In the following example, the mouse moves one inch down and one inch to the left when you click the *nervousMouse* button:

```
; nervousMouse::pushButton
method pushButton(var eventInfo Event)
var
  mouseP,
  newMouseP Point
endVar
mouseP = getMouseScreenPosition()
newMouseP = mouseP + Point(1440, 1440)
setMouseScreenPosition(newMouseP)   ; move pointer 1 inch down and
                                    ; 1 inch to the right
endMethod
```

getRegistryValue method

Retrieves a registry value.

Syntax

```
getRegistryValue ( const key String, const value String , const rootKey LongInt ) AnyType
```

Description

getRegistryValue retrieves data from a specified *key* and *value* in the registry. If **getRegistryValue** is successful, the registry value is returned as an AnyType; otherwise, it returns an empty string.

key is a path similar to a file path. However, wildcards are not expanded in the *key*. *key* cannot contain a single backslash and cannot be empty. Its size is limited to 65,534 bytes.

The *value* is a string that is limited to 65,534 bytes. *value* can contain backslashes and can be empty. *rootKey* is analogous to a directory drive. Set *rootKey* with the predefined RegistryKeyType Constants.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYDELETEREGISTRYKEY;OPAL_METH_SYENUMREGISTRYKEYS;OPAL_METH_SYENUMREGISTRYVALUENAMES;OPAL_METH_SYSEARCHREGISTRY;OPAL_METH_SYSETREGISTRYVALUE;',0,"Defaultoverview",)} Related Topics

getRegistryValue example

The following example retrieves the current ObjectPAL Level from the registry and displays it:

```
var
   strLevel   String
endvar

   strLevel = getRegistryValue( "Software\\Corel\\Paradox\\8.0\\Pdoxwin\\Properties", "Level",
      RegKeyCurrentUser )
   strLevel.view()
```

getUserLevel procedure

Returns your ObjectPAL level property setting (Advanced or Beginner).

Syntax

```
getUserLevel ( ) String
```

Description

getUserLevel returns Advanced or Beginner to specify your ObjectPAL level property setting. Use **setUserLevel** to change this setting.

Note

- The ObjectPAL level property setting *does not* affect code execution. The setting only affects the ObjectPAL language elements that are displayed in the user interface.

Example

**{button ,AL(`OPAL_TYPE_SYSTEM;INTRO_INTRO;OPAL_METH_SYSETUSERLEVEL;;;',0,"Defaultoverview",)
} Related Topics**

getUserLevel example

See the **setUserLevel** example.

helpOnHelp procedure

Displays information about using the Windows Help system and opens Help if necessary.

Syntax

```
helpOnHelp ( ) Logical
```

Description

helpOnHelp opens the WINHLP32.HLP file by default.

To open another Help file

1. Open the Help project file in a text editor.
2. Add a SetHelpOnFile macro to the [CONFIG] section, specifying the Help file you want to use in How to Use Help.
3. Compile the Help file.

The following macro, when placed in the [CONFIG] section of the Help project file changes the Help file, causes **helpOnHelp** to open:

```
[CONFIG]
SetHelpOnFile("howhelp.hlp")
```

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYHSCONT;OPAL_METH_SYHSINDEX;',0,"Defaultoverview",)} **Related Topics**

helpOnHelp example

The following example opens a Help file when you click Help, Help On Help from a custom menu:

```
method menuAction(var eventInfo MenuEvent)
   var
      siMenuChoice SmallInt
   endVar

   siMenuChoice = eventInfo.id()

   switch
      case siMenuChoice = UserMenu + MenuHelpOnHelp :
           helpOnHelp()
      ; Handle other cases here
   endSwitch

endmethod
```

helpQuit procedure

Notifies the Help application that it is no longer needed by the current application.

Syntax

```
helpQuit ( const helpFileName String ) Logical
```

Description

helpQuit notifies the Windows Help application (WINHELP.EXE) that the Help file *helpFileName* is no longer needed by the current Corel Paradox application. If the directory where *helpFileName* resides is not specified in the path, you must specify its full path. If no other applications require the Help application, Windows closes it.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYHONHELP;',0,"Defaultoverview",)} **Related Topics**

helpQuit example

The following example executes when you choose an item from a custom menu. If you click File, Close Form, **helpQuit** notifies the Help application that it is no longer needed and closes the current form.

```
method menuAction(var eventInfo MenuEvent)
   const
      ; Typically, menu choice constants are defined elsewhere,
      ; with the rest of the menu-building code. The following
      ; constant is defined here so the example will compile.
      kMyMenuFileCloseForm = 104
   endConst


   var
      siMenuChoice   SmallInt
      stHelpFileName String
   endVar

   siMenuChoice = eventInfo.id()
   stHelpFileName = "c:\\pdoxapps\\ordentry\\ordentry.hlp"


   switch
      case siMenuChoice = UserMenu + kMyMenuFileCloseForm :
           helpQuit(stHelpFileName) ; Tell Help we don't need it any more.
           close() ; Close the form.
   ; Handle other cases here
   endSwitch

endMethod
```

helpSetIndex procedure

Specifies what help file will be used as the Help contents (index).

Syntax

```
helpSetIndex ( const helpFileName String, const indexId LongInt ) Logical
```

Description

helpSetIndex specifies what help file will be used as the Help contents (index). This procedure instructs the Windows Help application (WINHELP.EXE) to use the topic in *helpFileName* (specified by *indexID*) as the Contents topic. If *helpFileName* does not reside in the directory specified in your path, you must specify the full path or the directory.

When you open a Help file, WinHelp displays the Contents topic by default. When you create a Help file, you specify the Contents topic using the Contents option in the [CONFIG] section of the Help project file. For example, when placed in the project file's [CONFIG] section, the following SetContents macro sets the Contents topic for a Help file to topic number 100 in CWH.HLP.

```
[CONFIG]
SetContents("cwh.hlp", 100)
```

If you do not use the SetContents option, the Contents topic is the first topic in the first file listed in the [FILES] section of the Help project file.

You can use **helpSetIndex** to specify a Contents topic from within an application.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYHSINDEX;',0,"Defaultoverview",)} **Related Topics**

helpSetIndex example

The following example sets the Contents topic for a Help file to topic number 100 in the file ORDENTRY.HLP:

```
method setHelpContents() Logical
   return helpSetIndex("c:\\pdoxapps\\ordentry\\ordentry.hlp", 100)
endMethod
```

helpShowContext procedure

Displays the Help topic specified by *helpId* in the file *helpFileName* .

Syntax

```
helpShowContext ( const helpFileName String, const helpId LongInt ) Logical
```

Description

helpShowContext instructs the Windows Help application to search *helpFileName* for the topic identified by *helpId*; and to display the topic. If the directory where *helpFileName* resides is not in your path, you must specify its full path.

In a Help source file, each topic is identified by a context ID. A context ID is a string defined by a # footnote. The context ID is mapped to an integer value in the [MAP] section of the Help project file (.HPJ). **helpShowContext** uses this mapped integer value to locate the Help topic.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYHSINDEX;OPAL_METH_SYHSTOPIC;',0,"Defaultoverview",)} Related Topics

helpShowContext example

The following example instructs the Windows Help application to display context-sensitive Help for the active object in a form. Assume that the form contains three buttons and two field objects. The code is attached to a button whose TabStop property is set to False. If the code is attached to a button whose TabStop property is set to True, the button becomes active when clicked.

```
helpButton::pushButton
const
; These integer values must also be listed
; in the [MAP] section of the Help project file.
   kNewOrdBtn   = LongInt(1020)
   kEditOrdBtn  = LongInt(1021)
   kDelOrdBtn   = LongInt(1022)
   kCustNameFld = LongInt(2020)
   kOrderNoFld  = LongInt (2021)
endConst


method pushButton(var eventInfo Event)


var
   stObjName,
   stHelpFileName String
   liContextId    LongInt
endVar


   stObjName = active.name ; Get the name of the active object.
   stHelpFileName = "c:\\pdoxapps\\ordentry\\ordentry.hlp"


   switch
      case stObjName = "newOrdBtn"    : liContextId = kNewOrdBtn
      case stObjName = "editOrdBtn"   : liContextId = kEditOrdBtn
      case stObjName = "delOrdBtn"    : liContextId = kDelOrdBtn
      case stObjName = "custNameFld"  : liContextId = kCustNameFld
      case stObjName = "orderNoFld"   : liContextId = kOrderNoFld
   endSwitch


   if not helpShowContext(stHelpFileName, liContextId) then
      errorShow("Could not display Help topic.")
   endIf


endMethod
```

helpShowIndex procedure

Displays the contents topic (index) of a specified Help file.

Syntax

```
helpShowIndex ( const helpFileName String ) Logical
```

Description

helpShowIndex instructs the Windows Help application (WINHELP.EXE) to display the Contents topic (*index*) in the Help file specified by *helpFileName*. If the directory where *helpFileName* resides is not on your path, you must specify its full path.

When you open a Help file, WinHelp displays the Contents topic by default. When you create a Help file, you specify the Contents topic using the Contents option in the [CONFIG] section of the Help project file. For example, when placed in the project file's [CONFIG] section, the following SetContents macro sets the Contents topic for a Help file to topic number 100 in CWH.HLP.

```
[CONFIG]
SetContents("cwh.hlp", 100)
```

If you do not use the SetContents option, the Contents topic is the first topic in the first file listed in the [FILES] section of the Help project file.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYHSCONT;OPAL_METH_SYHSTOPIC;',0,"Defaultoverview",)} Related Topics

helpShowIndex example

The following example executes when you choose an item from a custom menu. If you click Help, Contents, **helpShowIndex** instructs the Help application to display the Contents topic for the specified Help file.

```
method menuAction(var eventInfo MenuEvent)
   const
      ; Typically, menu choice constants are defined elsewhere,
      ; with the rest of the menu-building code. The following
      ; constant is defined here so the example will compile.
      kMyMenuHelpContents = 501
   endConst


   var
      siMenuChoice   SmallInt
      stHelpFileName String
   endVar


   siMenuChoice = eventInfo.id()
   stHelpFileName = "c:\\pdoxapps\\ordentry\\ordentry.hlp"


   switch
      case siMenuChoice = UserMenu + kMyMenuHelpContents :
           helpShowIndex(stHelpFileName) ; Display the Contents topic.
      ; Handle other cases here
   endSwitch


endMethod
```

helpShowTopic procedure

Displays help for a specified context ID.

Syntax

helpShowTopic (const ***helpFileName*** String, const ***topicKey*** String **)** Logical

Description

helpShowTopic instructs the Windows Help application to search the file *helpFileName* for the topic associated with *topicKey*, and to display the topic. If the directory where *helpFileName* resides is not on your path, you must specify its full path. *topicKey* must match a keyword defined by a K footnote in the Help source file. If *topicKey* does not match a keyword, the search fails and the Windows Help application displays an error message.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYHSKEY;OPAL_METH_SYHSCONT;OPAL_METH_SYHSINDEX;',0,"Defaultoverview",)} Related Topics

helpShowTopic example

The following example prompts you to type a word or phrase and then searches for the text in the specified Help file:

```
method pushButton(var eventInfo Event)
   var
      stHelpFileName,
      stTopicKey,
      stPromptText    String
   endVar


   stHelpFileName = "c:\\pdoxapps\\ordentry\\ordEntry.hlp"
   stPromptText   = "Enter a word or phrase here."
   stTopicKey     = stPromptText


   stTopicKey.view("Enter text to search for.")
   if stTopicKey <> stPromptText then
      helpShowTopic(stHelpFileName, stTopicKey)
   endIf
endMethod
```

helpShowTopicInKeywordTable procedure

Displays Help for a topic identified by a keyword in an alternate keyword table.

Syntax

helpShowTopicInKeywordTable (const ***helpFileName*** String, const ***keyTableLetter*** String, const ***topicKey*** String **)** Logical

Description

helpShowTopicInKeywordTable instructs the Windows Help application to search the file *helpFileName* for the topic associated with *keyTableLetter* and *topicKey*, and to display the topic. If the directory where *helpFileName* resides is not in your path, you must specify its full path. The value of *keyTableLetter* must match a multi-key index specified in the [OPTIONS] section of the Help project file. For example, if a Help project file includes the following code, assign L to *keyTableLetter*.

```
[OPTIONS]
MULTIKEY=L
```

The value of *topicKey* must match a keyword defined using a multi-key index footnote in the Help source file. If *topicKey* does not match, the search fails and the Windows Help application displays an error message.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYHSTOPIC;OPAL_METH_SYHSCONT;OPAL_METH_SYHSINDEX;',0,"Defaultoverview",)} **Related Topics**

helpShowTopicInKeywordTable example

The following example prompts you to type COREL PARADOX or dBASE and then searches for field types in the keyword table of the specified Help file. Assume that an application is handling a user's request for Help on the topic field types.

```
method pushButton(var eventInfo Event)
   var
      stHelpFileName,
      stPromptText,
      stUserChoice,
      stTopicKey,
      stKeyTableLetter    String
   endVar


   stHelpFileName   = "c:\\pdoxapps\\ordentry\\ordEntry.hlp"
   stPromptText     = "Enter COREL PARADOX or dBASE here."
   stUserChoice     = stPromptText
   stTopicKey       = "field types"


   stUserChoice.view("Do you want Corel Paradox Help or dBASE Help?")
   if stUserChoice <> stPromptText then
      switch
         case stUserChoice = "COREL PARADOX" : stKeyTableLetter = "P"
         case stUserChoice = "dBASE"   : stKeyTableLetter = "D"
         otherwise : return
      endSwitch


      helpShowTopicInKeywordTable(stHelpFileName, stKeyTableLetter, stTopicKey)
   endIf
endMethod
```

isErrorTrapOnWarnings procedure

Reports whether this session handles warning errors as critical errors.

Syntax

```
isErrorTrapOnWarnings ( ) Logical
```

Description

isErrorTrapOnWarnings reports whether this session handles warning errors as critical errors. This method returns True if the active session treats warning errors as critical errors; otherwise, it returns False.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYERRORTRAPONWARNINGS;',0,"Defaultoverview",)}

Related Topics

isErrorTrapOnWarnings example

The following example uses the **pushButton** method for *btnToggleWarning* to toggle between critical and non-critical warning errors:

```
; btnToggleWarning :: pushButton
method pushButton(var eventInfo Event)
  errorTrapOnWarnings(not isErrorTrapOnWarnings())
  msgInfo("Warning errors are critical", isErrorTrapOnWarnings())
endmethod
```

isMousePersistent method

Reports if mouse persistence is turned on.

Syntax

isMousePersistent ( ) Logical

Description

isMousePersistent reports if mouse persistence is on. **isMousePersistent** returns True if mouse persistence is turned on, and False if mouse persistence is turned off. To set mouse persistence, use **setMouseShape** or **setMouseShapeFromFile**.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYSMPOS;OPAL_METH_SYSMSHAP;OPAL_METH_SYSETMOUSESHAPEFROMFILE;',0,"Defaultoverview",)} Related Topics

isMousePersistent example

In the following example, a form has two buttons: *btnNonPersistent* and *btnPersistent*. The **pushButton** method for each button uses setMouseShape() to set the mouse shape of the cursor. The first button has mouse persistence turned off, and the second button had mouse persistence turned on. The second button, *btnPersistent*, also contains a mouseEnter method which uses isMousePersistent() to evaluate the persistency of the cursor and revert it to its original state. When the first button is pressed, the pointer changes. However, when the cursor moves off the button, it reverts to its original setting. When the second button is pressed, the cursor changes and remains unmodified until the cursor moves back over the second button. This triggers the mouseEnter method of the second button and reverts the cursor back to its original state.

The following code is attached to the **pushButton** method for *btnNonPersistent*:

```
; btnNonPersistent::pushButton
method pushButton(var eventInfo Event)
   ;// Set the shape to MouseWait and persistence to False
   setMouseShape(MouseWait,FALSE)
endMethod
```

The following code is attached to the **pushButton** method for *btnPersistent*:

```
; btnPersistent::pushButton
method pushButton(var eventInfo Event)
   ;// Set the shape to MouseWait and persistence to TRUE
   setMouseShape(MouseWait,TRUE)
endMethod
```

The following code is attached to the **mouseEnter** method for *btnPersistent*:

```
; btnPersistent::mouseEnter
method mouseEnterpushButton(var eventInfo MouseEvent)
   if isMousePersistent() then
      ;// If it's persistent, set it back to the arrow cursor
      setMouseShape(MouseArrow,FALSE)
   endIf
endMethod
```

message procedure

Displays a message composed of up to six strings in the status line.

Syntax

```
message ( const message String [ , const message String ] * )
```

Description

message displays a message composed of up to six strings in the status line.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYMINF;OPAL_METH_SYMQUE;OPAL_METH_SYMSTP;',0,"Defaultoverview",)} **Related Topics**

message example

The following example writes a message to the status line:

```
; showMessage::pushButton
method pushButton(var eventInfo Event)
var
  lastName, firstName String
endVar
lastName = "Corel"
firstName = "Frank"
message("Hello, my name is ", firstName, " ", lastName, ".")
endMethod
```

msgAbortRetryIgnore procedure

Displays a dialog box containing a message and the Abort, Retry, and Ignore buttons.

Syntax

```
msgAbortRetryIgnore ( const caption String, const text String ) String
```

Description

msgAbortRetryIgnore displays a three-button dialog box, where *caption* specifies the text in the Title Bar and *text* specifies the message. The return value is a mixed upper and lowercase string, that corresponds to the button you click.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYMRCA;OPAL_METH_SYMYNC;',0,"Defaultoverview",)}

Related Topics

msgAbortRetryIgnore example

The following example uses the *showAbortRetryIgnore* button to warn you that an operation may take a long time and asks you whether to Abort, Retry, or Ignore:

```
; showAbortRetryIgnore::pushButton
method pushButton(var eventInfo Event)
var
  doThis String
endVar
doThis = msgAbortRetryIgnore("Note", "This may take a long time.
Do you want to stop?")  ; This message spans 2 lines.


doThis.view() ; Display your choice.


; Display a message based on your choice.
switch
   case doThis = "Abort"  : message("Aborting operation.")
   case doThis = "Retry"  : message("Retrying operation.")
   case doThis = "Ignore" : message("Ignoring problem.")
endSwitch
endMethod
```

msgInfo procedure

Displays a one-button dialog box containing the information icon, a caption and message, and an OK button.

Syntax

```
msgInfo ( const caption String, const text String )
```

Description

msgInfo displays a one-button dialog box containing the information icon, a caption and message, and an OK button. *caption* is displayed in the Title Bar, and *text* is displayed in the box. Click OK or press ESC to close the dialog box. This procedure does not return a value.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYMQUE;OPAL_METH_SYMSTP;',0,"Defaultoverview",)}

Related Topics

msgInfo example

The following example uses the **msgInfo** method displays a message:

```
; showMsgInfo::pushButton
method pushButton(var eventInfo Event)
msgInfo("Trivia", "The capital of Oregon is Salem.")
endMethod
```

msgQuestion procedure

Displays a dialog box containing a caption and message, a question mark icon, and Yes and No buttons.

Syntax

```
msgQuestion ( const caption String, const text String ) String
```

Description

msgQuestion displays a dialog box containing a caption and message, a question mark icon, and Yes and No buttons. It displays *caption* in the Title Bar, and *text* in the box itself. This procedure returns your selection (Yes or No) in mixed upper and lowercase.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYMINF;OPAL_METH_SYMSTP;',0,"Defaultoverview",)}

Related Topics

msgQuestion example

The following example asks you whether to change the desktop title. If you choose Yes, the desktop title is changed and then restored.

```
; showMsgQuestion::pushButton
method pushButton(var eventInfo Event)
var
  userChoice String
  thisApp    Application
endVar
userChoice = msgQuestion("Confirm", "Are you sure you want to
change the title to 'Custom Application'?")
switch
   case userChoice = "Yes" :
     thisApp.setTitle("Custom Application")  ; Change desktop title.
     sleep(2000)                             ; Pause.
     thisApp.setTitle("Corel Paradox for Windows") ; Restore it.
   case userChoice = "No"  :
     message("Application title not changed.")
endSwitch
endMethod
```

msgRetryCancel procedure

Displays a dialog box containing a caption, a message, and the Retry and Cancel buttons.

Syntax

```
msgRetryCancel ( const caption String, const text String ) String
```

Description

msgRetryCancel displays a dialog box containing a caption, a message, and the Retry and Cancel buttons. The argument *caption* specifies the text in the dialog box's Title Bar. *text* specifies the message displayed. **msgRetryCancel** returns your selection (Retry or Cancel). If you press ESC or select Close, returns Cancel. Values are returned in mixed upper and lowercase.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYMYNC;OPAL_METH_SYMARI;',0,"Defaultoverview",)}

Related Topics

msgRetryCancel example

The following example poses a question and confirms your response on the status line:

```
; showMsgRetryCancel::pushButton
method pushButton(var eventInfo Event)
var
  confirm String
endVar
confirm = msgRetryCancel("Dilemma", "What will you do?")
switch
  case confirm = "Retry"  : message("Retrying.")
  case confirm = "Cancel" : message("Giving up.")
endSwitch
endMethod
```

msgStop procedure

Displays a dialog box containing a stop sign icon, a caption and message, and an OK button.

Syntax

```
msgStop ( const caption String, const text String )
```

Description

msgStop displays a dialog box containing a stop sign icon, a caption and message, and an OK button. It displays *caption* in the Title Bar, and *text* and a Stop icon in the box itself. Click OK or press ESC to close the dialog box. This procedure does not return a value.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYMINF;OPAL_METH_SYMQUE;',0,"Defaultoverview",)}

Related Topics

msgStop example

The following example uses the **pushButton** method for *showMsgStop* to alert you to a potentially dangerous action:

```
; showMsgStop::pushButton
method pushButton(var eventInfo Event)
msgStop("Stop!", "If you do that, changes to the form will not be saved.")
endMethod
```

msgYesNoCancel procedure

Displays a dialog box containing a caption, a message and the Yes, No, and Cancel buttons.

Syntax

```
msgYesNoCancel ( const caption String, const text String ) String
```

Description

msgYesNoCancel displays a dialog box containing a caption, a message and the Yes, No, and Cancel buttons. The argument *caption* specifies the text in the dialog box's Title Bar. *text* specifies the message displayed. **msgYesNoCancel** returns your selection (Yes, No or Cancel) in mixed upper and lowercase. If you press ESC or select Close, this procedure returns Cancel.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYMRCA;OPAL_METH_SYMARI;',0,"Defaultoverview",)}

Related Topics

msgYesNoCancel example

The following example uses **msgYesNoCancel** to ask you whether to save the data before quitting, to discard the data, or to cancel the quit the operation:

```
; showMsgYesNoCancel::pushButton
method pushButton(var eventInfo Event)
var
  theChoice String
endVar
theChoice = msgYesNoCancel("Quit", "Save data before quitting?")
switch
   case theChoice = "Yes"    : message("Saving data.")
   case theChoice = "No"     : message("Discarding data.")
   case theChoice = "Cancel" : message("Remaining in application.")
endSwitch
endMethod
```

pixelsToTwips procedure

Converts the screen coordinates from pixels to twips.

Syntax

```
pixelsToTwips ( const pixels Point ) Point
```

Description

pixelsToTwips converts the screen coordinates from pixels to twips.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYTWIPSTOPIXELS;',0,"Defaultoverview",)} Related Topics

pixelsToTwips example

The following example uses the object variable self to show the position of the button in twips and in pixels. This code displays the screen resolution in pixels opens a window in the center of the display.

```
; convertTwipsPixels::pushButton
method pushButton(var eventInfo Event)
var
  selfP,
  sysTwips  Point
  thisSys   DynArray[] AnyType
  x, y      SmallInt
  custForm  Form
endVar
selfP = self.Position
selfP.view("Position of this button in twips")
selfP = twipsToPixels(selfP)
selfP.view("Position of this button in pixels")
; open a 2" by 2" form exactly in the center of the screen
sysInfo(thisSys)                ; fill a dynamic array with system information
sysTwips = Point(thisSys["FullWidth"], thisSys["FullHeight"])
sysTwips = pixelsToTwips(sysTwips)
x = int(sysTwips.x()/2) - 1440  ; calculate x-coordinate 1 inch left of center
y = int(sysTwips.y()/2) - 1440  ; calculate y-coordinate 1 inch above center
custForm.open("Customer.fsl", WinStyleDefault, x, y, 2880, 2880)

endMethod
```

play procedure

Plays a standalone script.

Syntax

```
play ( const scriptName String ) AnyType
```

Description

play executes *scriptName* to play a standalone script. To return a value from a script, call **formReturn** from within the script.

For more information, refer to the Script type.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYEXEC;OPAL_TYPE_SCRIPT;',0,"Defaultoverview",)}

Related Topics

play example

The following example plays a script called TESTSCR.SSL, which resides in the working directory:

```
; playAScript::pushButton
method pushButton(var eventInfo Event)
play("Testscr.ssl")
endMethod
```

printerGetInfo procedure

Retrieves information about the printer installed on your system.

Syntax

```
printerGetInfo ( var printInfo PrinterInfo ) Logical
```

Description

printerGetInfo assigns printer information to *printInfo*, a record that you declare using a special ObjectPAL data type named PrinterInfo. The following table displays the structure of PrinterInfo:

Field	Type	Description
DriverName	String	Name of the printer driver (e.g., PSCRIPT.DRV)
DeviceName	String	Name that identifies the printer type (e.g., Apple LaserWriter Plus)
PortName	String	Name of the printer port (e.g., LPT1)
DefaultPrinter	Logical	Determines whether the current printer is the default

This procedure returns True if successful; otherwise, it returns False.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYENUMPRINTERS;OPAL_METH_SYPRINTERGETOPTIONS;OPAL_METH_SYPRINTERSETCURRENT;OPAL_METH_SYPRINTERSETOPTIONS;',0,"Defaultoverview",)}

Related Topics

printerGetInfo example

See the **printerSetOptions** example.

printerGetOptions procedure

Retrieves information about your system printer's settings.

Syntax

```
1. printerGetOptions ( var printOptions PrinterOptionInfo ) Logical
2. printerGetOptions ( var printerInfo DynArray[] AnyType ) Logical
```

Description

printerGetOptions assigns printer information to *printInfo*. *PrintInfo* is a variable you declare as an ObjectPAL record with a predefined structure called PrinterOptionInfo.

printerGetOptions assigns printer information to *printInfo*, a record you declare as an ObjectPAL data type PrinterOptionInfo.

Syntax 2 fills an array named *printerInfo* with supported print options.

This procedure returns True if successful; otherwise, it returns False.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYENUMPRINTERS;OPAL_METH_SYPRINTERGETINFO;OPAL_METH_SYPRINTERSETCURRENT;OPAL_METH_SYPRINTERSETOPTIONS;',0,"Defaultoverview",)} Related Topics

printerGetOptions example

The following example sets the current printer settings and determines whether the printer is using a large format paper source:

```
method pushButton(var eventInfo Event)
   var
      recUserOptions,
      recMyOptions   PrinterOptionInfo
   endVar


   ; Get the current printer settings.
   printerGetOptions(recUserOptions)
   if recUserOptions.DefaultSource = prnLargeFmt then
      return
   endIf


   ; Specify new printer settings. prnLargeFmt is a PrintSources constant.
   recMyOptions.DefaultSource = prnLargeFmt


   if printerSetOptions(recMyOptions) then
      message("Printer setup complete.")
   else
      errorShow()
   endIf
endMethod
```

PrinterOptionInfo record structure

Field	Type	Description
Orientation	LongInt	Paper orientation (portrait or landscape). Use a PrinterOrientation constant to test the value.
PaperSize	LongInt	Paper size. Use a PrinterSizes constant to test the value.
PaperWidth	LongInt	Custom paper width in twips (maximum of 64K twips). This value is converted internally to the tenths of a millimeter required by Windows.
PaperLength	LongInt	Custom paper length in twips (maximum of 64K twips). This value is converted internally to the tenths of a millimeter required by Windows.
Scale	LongInt	Scaling factor in percent. A scale value of 50 reduces the original to one-half its size. A value of 200 increases the original to twice its size. Scaling only applies to printers that support scaling for all functions, graphics, and fonts (e.g., Postscript printers and the Microsoft Windows Printing System).
Copies	LongInt	Number of copies for the printer to make. The Copies option works only with page printers (e.g., laser printers) where the full page can be held in printer memory. Some printer drivers support this feature on printers that cannot do full page printing. The Copies setting is equivalent to unchecking the Collate button in the Print File dialog box. Output is not collated. This operation is faster than repeatedly sending the full document to the printer, but requires hand sorting at completion.
DefaultSource	LongInt	Bin, tray, or feeder used by the default printer. Use a PrintSources constant to test the value.
PrintQuality	LongInt	Higher print qualities are used for final output, and lower print qualities for draft output. Lower quality prints differ significantly from the preview appearance of the document. Use a PrintQuality constant to test the value.
Color	LongInt	Sets color printers to color or monochrome printing. Monochrome printing is usually faster. Use a PrintColor constant to test the value.
Duplex	LongInt	Double-sided printing. Some printer drivers can support double-sided printing on otherwise single-sided printers by making two passes over the document. Use a PrintDuplex constant to test the value.

printerSetCurrent procedure

Sets the active printer on your system.

Syntax

printerSetCurrent (*printerInfo* String) Logical

Description

printerSetCurrent sets the active printer on your system. The argument *printerInfo* specifies the printer name, driver name, and printer port (separated by commas). For example, if the printer name is Postscript Printer, the driver is PSCRIPT.DRV, and the port is LPT1, the following code applies:

```
PostScript Printer,pscript,LPT1:
```

This procedure returns True if successful; otherwise, it returns False.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYENUMPRINTERS;OPAL_METH_SYPRINTERGETINFO;OPAL_METH_SYPRINTERGETOPTIONS;OPAL_METH_SYPRINTERSETOPTIONS;',0,"Defaultoverview",)} Related Topics

printerSetCurrent example

See the **enumPrinters** example.

printerSetOptions procedure

Specifies settings for your system printer.

Syntax

```
1. printerSetOptions ( PrintOptions PrinterOptionInfo ) Logical
2. printerSetOptions ( var printerInfo DynArray[] AnyType [const overRide Logical] ) Logical
```

Description

printerSetOptions specifies settings for your system printer. *printerSettings* is a record of the special ObjectPAL data type PrinterOptionInfo that you must declare. You don't have to specify values for each field in a PrinterOptionInfo record. The printer substitutes its current setting for any value you don't specify.

Syntax 2 uses an array named *printerInfo* (obtained with **printerGetOptions**) to send the printer settings for only those options that the printer supports. The optional *overRide* argument tells **printerSetOptions** to override printer settings specified in the Form or Report level.

printerSetOptions returns True if successful; otherwise, it returns False. If you specify a value that doesn't apply to your printer, this method returns False.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYENUMPRINTERS;OPAL_METH_SYPRINTERGETINFO;OPAL_METH_SYPRINTERGETOPTIONS;OPAL_METH_SYPRINTERSETCURRENT;',0,"Defaultoverview",)} Related Topics

printerSetOptions example

The following example prompts you to specify the number of copies of a report to print, sets up the printer, and prints the copies:

```
method pushButton(var eventInfo Event)
   var
      siNCopies   SmallInt
      stPrompt    String
      prnOptions  PrinterOptionInfo
      reOrders    Report
   endVar


   siNCopies = 0
   stPrompt  = "Print how many copies?"


   siNCopies.view(stPrompt)
   if siNCopies > 0 then
      prnOptions.Copies = siNCopies
   else
      return
   endIf


; Use constant to specify lower paper tray.
   prnOptions.DefaultSource = prnLower


; Use constant to specify landscape (long) orientation.
   prnOptions.Orientation = prnLandscape


; Use constant to specify high quality print.
   prnOptions.PrintQuality = prnHigh


   if printerSetOptions(prnOptions) then
      reOrders.print("orders")
   else
      errorShow("Could not set printer options.")
   endIf


endMethod
```

projectViewerClose procedure

Closes the Project Viewer window.

Syntax

```
projectViewerClose ( ) Logical
```

Description

projectViewerClose closes the Project Viewer window. This procedure returns True if successful; otherwise, it returns False.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYPROJECTVIEWERISOPEN;OPAL_METH_SYPROJECTVIEWEROPEN;',0,"Defaultoverview",)} Related Topics

projectViewerClose example

The following example calls **projectViewerIsOpen** to determine whether the Project Viewer window is open. If the Project viewer is open, this code closes it.

```
method open(var eventInfo Event)
   if eventInfo.isPreFilter() then
      ;// This code executes for each object on the form:

   else
      ;// This code executes only for the form:
      if projectViewerIsOpen() then
         projectViewerClose()
      endIf
   endIf
endMethod
```

projectViewerIsOpen procedure

Tells whether the Project Viewer window is open.

Syntax

```
projectViewerIsOpen ( ) Logical
```

Description

projectViewerIsOpen determines whether the Project Viewer window is open. This procedure returns True if the Project Viewer window is open; otherwise, it returns False.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYPROJECTVIEWERCLOSE;OPAL_METH_SYPROJECTVIEWEROPEN;',0,"Defaultoverview",)} Related Topics

projectViewerIsOpen example

See the **projectViewerClose** example.

projectViewerOpen procedure

Opens the Project Viewer window.

Syntax

```
projectViewerOpen ( ) Logical
```

Description

projectViewerOpen opens the Project Viewer window. This procedure returns True if successful; otherwise, it returns False.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYPROJECTVIEWERCLOSE;OPAL_METH_SYPROJECTVIEWERISOPEN;',0,"Defaultoverview",)} **Related Topics**

projectViewerOpen example

The following example calls **projectViewerIsOpen** to determine whether the Project Viewer window is open. If the Project viewer is open, this code closes it.

```
method open(var eventInfo Event)
   if eventInfo.isPreFilter() then
      ;// This code executes for each object on the form:

   else
      ;// This code executes only for the form:
      if not projectViewerIsOpen() then
         projectViewerOpen()
      endIf
   endIf
endMethod
```

readEnvironmentString procedure

Reads an item from the Corel Paradox copy of the DOS environment.

Syntax

readEnvironmentString (const ***key*** String) String

Description

readEnvironmentString returns a string containing information about the DOS environment variable specified by *key*. When you launch Corel Paradox it makes a copy of the DOS environment. **readEnvironmentString** reads that copy and compiles information in a string. Changes made to DOS environment variables after Corel Paradox is launched are not read by this procedure.

The DOS command SET assigns values to the environment variables. These values control the appearance and function of DOS and some batch files. Commonly used environment variables include PATH, PROMPT, and COMSPEC. For more information, see the SET command your DOS manuals, especially .

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYRPROF;OPAL_METH_SYWRITEENVIRONMENTSTRING;',0,"Defaultoverview",)} Related Topics

readEnvironmentString example

The following example uses **readEnvironmentString** to retrieve the value of the PATH environment variable. The code then uses **writeEnvironmentString** to change it.

```
; changeEnvironmentStr::pushButton
method pushButton(var eventInfo Event)
var
   fs              FileSystem
   thePath, myDir  String
   pathArr Array[] String
endVar
; fs.getDir() currently returns some high-ANSI char--not a meaningful string
myDir = getaliaspath(fs.getDir())       ; get the current directory
myDir.view("Current directory")
thePath = readEnvironmentString("PATH") ; read the path environment var
thePath.breakApart(pathArr, ";")        ; break on semicolon
pathArr.view("An array of paths")       ; view the results
if NOT pathArr.contains(myDir) then     ; if current dir not in path
   msgInfo("FYI", "Adding current directory to path.")
   writeEnvironmentString("PATH", thePath + ";" + myDir)  ; add it
endIf
thePath = readEnvironmentString("PATH") ; read the changed environment var
thePath.view()
thePath.breakApart(pathArr, ";")        ; break it up
pathArr.view("An array of paths")       ; view the results
endMethod
```

readProfileString procedure

Returns a value from a specified section of a file.

Syntax

```
readProfileString ( const fileName String, const section String, const key String ) String
```

Description

readProfileString returns a value from a specified section of a file. By default this procedure searches the WINDOWS directory. You can also use this method to read your WIN.INI file, so *fileName* would be WIN.INI.

Each section header in WIN.INI is bounded by square brackets on a separate line (e.g., [windows]). To specify a *section,* omit the brackets (e.g., use windows). In each section, a value marker is followed by an equal sign (e.g., Beep =). The equal sign is not required when you specify the value of *key*.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYRENV;',0,"Defaultoverview",)} Related Topics

readProfileString example

The following example uses **readProfileString** to retrieve the setting for the Windows beep, and **writeProfileString** to change the setting:

```
; changeProfileStr::pushButton
method pushButton(var eventInfo Event)
var
   myBeep String
   winDir String
endVar
winDir = windowsDir()
myBeep = readProfileString(winDir + "\\win.ini", "windows", "Beep")
msgInfo("Beep?", myBeep) ; displays yes or no, depending on user's settings
if myBeep <> "yes" then
  msgInfo("Alert", "Changing profile string for Beep to yes.")
  writeProfileString(winDir + "\\win.ini", "windows", "Beep", "yes")
  beep()
else
  msgInfo("Alert", "Changing profile string for Beep to no.")
  writeProfileString(winDir + "\\win.ini", "windows", "Beep", "no")
  beep()
endIf
endMethod
```

resourceInfo procedure

Lists the system resources.

Syntax

```
resourceInfo ( var info DynArray[ ] AnyType )
```

Description

resourceInfo writes system resource data to *info*. *Info* is a dynamic array (DynArray) that you declare and pass as an argument.

The following table displays the information returned in *info*:

Index	Definition
DiskAvail	Available disk space on the current drive
DiskTotal	Total disk space on the current drive
FreeGdiResources	Percentage of free Windows GDI resources. This item is not supported in the 32-bit Windows environment.)
FreeSpace	Free Windows memory
FreeSystemResources	Percentage of free Windows system resources. This item is not supported in the 32-bit Windows environment.
FreeUserResources	Percentage of free Windows user resources. This item is not supported in the 32-bit Windows environment.
InternalVersion	Corel Paradox internal Borland Database Engine (BDE) version
MemoryLoad	Percent of memory in use
MemPhysicalTotal	Total physical memory
MemPhysicalFree	Available physical memory
MemPageFileTotal	Total page/file memory
MemPageFileFree	Available page/file memory
MemVirtualTotal	Total virtual memory
MemVirtualFree	Available virtual memory

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYENUMENVIRONMENTSTRINGS;',0,"Defaultoverview",)}

Related Topics

resourceInfo example

The following example writes resource information to a dynamic array named *dyn* and then displays *dyn* in a View dialog box:

```
; btnResourceInfo::pushButton
method pushButton(var eventInfo Event)
  var
    dynResources  Dynarray[] String
  endVar


  resourceInfo(dynResources)
  dynResources.view()
endmethod
```

runExpert procedure

Runs a registered Corel Paradox expert.

Syntax

```
runExpert ( const expertType String, const expertName String )
```

Description

runExpert runs a registered Corel Paradox expert. The *expertName* argument specifies which expert to run. The *expertType* parameter determines the type of experts to list. ObjectPAL provides ExpertTypes constants for this purpose.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYENUMEXPERTS;',0,"Defaultoverview",)} **Related Topics**

runExpert example

The following example enumerates the available experts, and runs expertForm if it is available:

```
method pushButton(var eventInfo Event)
var
   da dynarray[] anytype
   expert string
endvar


expertForm = "Form"
enumExperts(  "Document", da )


if da.contains( expertForm ) then
   runExpert( "Document", expertForm )
else
   msgStop( "Error", "Unable to run the expert :" + expertForm )
endif
endmethod
```

searchRegistry procedure

Searches the registry for a specified value.

Syntax

```
searchRegistry ( const key String, const searchStr String, const rootKey LongInt, const
searchMode LongInt, const inMem TCursor ) Logical
```

Description

searchRegistry searches the registry string data types for the value in *searchStr*. Searches performed by **searchRegistry** are case insensitive and the results are placed in *inMem*, an in-memory TCursor. **searchRegistry** returns True if successful; otherwise, it returns False.

key is entered as a path similar to a file path. If *key* is not blank, the search begins at the specified path; otherwise, it starts from the *rootKey*. *searchStr* is the value of the object you want to locate. **searchRegistry** only searches strings, and not registry DWORD or Binary types. If *searchStr* is blank, **searchRegistry** returns an error. Set *rootKey* with the predefined RegistryKeyType Constants, or it can be set to zero. If *rootKey* is zero, then all *rootKey*s are searched.

searchMode specifies the registry objects you want to search in the registry. Registry objects include keys, value names, and data. The following table describes the *searchMode* flags:

searchMode	Registry objects searched
0	All
1	Keys
2	Value names
3	Data
4	Keys and value names
5	Keys and Data
6	Value names and Data

The *inMem* TCursor has three fields that are limited to A255. The values in these fields are truncated if the key returned is greater than 255 characters. **searchRegistry** returns a warning if the field limit is reached. The following table displays the structure of *inMem*:

Field	Type	Description
RegistryType	A12	Registry object type (Key, ValueName or Data)
RootKeyConst	A25	Predefined *rootKey* constant as a string
KeyPath	A255	Full path of the key
ValueName	A255	Full value name
Data	A255	Full Data

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYDELETEREGISTRYKEY;OPAL_METH_SYENUMREGISTRYKEYS;OPAL_METH_SYENUMREGISTRYVALUENAMES;OPAL_METH_SYGETREGISTRYVALUE;OPAL_METH_SYSETREGISTRYVALUE;',0,"Defaultoverview",)} **Related Topics**

searchRegistry example

The following example searches the registry for all keys containing the string Corel. The results are displayed in a TableView window.

```
;// Search the registry for keys that have the string "Corel" in them, and write
;// the results to a table
var
   tc TCursor
endVar
searchRegistry( "", "Corel", 0, 1, tc )  ; Search the registry
                                           ; for keys that have
                                           ; "Corel" in them
if NOT tc.isEmpty() then
   tc.instantiateView("keytab.db")    ; write the results to a table
endif
tc.close()
```

The following example searches the entire registry for keys containing the word Pdoxwin. The results are displayed in a TableView window.

```
var
   tc   Tcursor
   tv   TableView
endvar


searchRegistry( "", "Pdoxwin", 0, 1, tc )
tc.instantiateView( ":priv:keysreg" )
tv.open( ":priv:keysreg" )
tv.wait()
tv.close()
```

sendKeys procedure

Sends one or more keystrokes to the active window.

Syntax

```
sendkeys ( const keyText String [ , const wait Logical ] ) Logical
```

Description

sendKeys sends one or more keystrokes to the active window as if they had been entered at the keyboard. The active window does not have to be Corel Paradox. The argument *keyText* specifies the keystrokes to send. *wait* (optional) specifies whether to continue executing keystroke sequences in the message loop without waiting. **sendKeys** returns False if an error it from sending the keys. **errorCode** returns one of the following messages:

Error code	Error message
peskMissingCloseBrace	Missing closing brace
peskInvalidKey	The key name is not correct
peskMissingCloseParen	Missing closing parentheses
peskInvalidCount	The repeat count is not correct
peskStringTooLong	The keys string is too long
peskCantInstallHook	Could not install Windows journal hook

Note

- **sendKeys** can only send keystrokes to Microsoft Windows applications. It cannot send the Print Screen (PRINT SCRN) key to any application.

The keyText argument

Each key is represented by one or more lowercase characters. To represent the letter A, use "a" for *keyText*. To represent more than one character, string them together. To send the letters a, b, and c, use "abc" for *keyText*. The plus sign (+), caret (^), percent sign (%), tilde (~), and parentheses ( ) have special meanings to **sendKeys**. To specify one of these characters, enclose it inside braces. To specify the plus sign, use{+}. To send brace characters, enclose each brace in braces: {{} and {}}.

To specify non-printing characters (such as ENTER or TAB) and keys that represent actions rather than characters, use the following codes:

Key	Codes
BACKSPACE	{backspace}, {bs}, {bksp}, {vk_back}
BREAK	{break}, {vk_break}
CAPS LOCK	{capslock}, {vk_captial}
CLEAR	{clear}, {vk_clear}
DEL	{delete}, {del}, {vk_delete}
Down Arrow	{down}, {vk_down}
END	{end}, {vk_end}
ENTER	{enter}, {return}, {vk_return} (the character ~)
ESC	{escape}, {esc}, {vk_escape}
HELP	{help}, {vk_help}
HOME	{home}, {vk_home}
INS	{insert}, {vk_insert}
Left Arrow	{left}, {vk_left}
NUM LOCK	{numlock}, {vk_numlock}
PAGE DOWN	{pgdn}, {vk_next}
PAGE UP	{pgup}, {vk_prior}
PRINT SCRN	{prtsc}, {vk_snapshot}
Right Arrow	{right}, {vk_right}
SCROLL LOCK	{scrolllock}, {vk_scroll}
SPACEBAR	{vk_space}
TAB	{tab}, {vk_tab}

Up Arrow	{up}, {vk_up}
F1	{F1}, {vk_F1}
F2	{F2}, {vk_F2}
F3	{F3}, {vk_F3}
F4	{F4}, {vk_F4}
F5	{F5}, {vk_F5}
F6	{F6}, {vk_F6}
F7	{F7}, {vk_F7}
F8	{F8}, {vk_F8}
F9	{F9}, {vk_F9}
F10	{F10}, {vk_F10}
F11	{F11}, {vk_F11}
F12	{F12}, {vk_F12}
F13	{F13}, {vk_F13}
F14	{F14}, {vk_F14}
F15	{F15}, {vk_F15}
F16	{F16}, {vk_F16}

The ~ character represents the ENTER key. For example, `sendKeys("abc~")` types the letters abc and the carriage return.

To specify keys combined with SHIFT, CTRL, and ALT, precede the regular key code with one or more of the following codes:

Key	Code
SHIFT	+
CTRL	^
ALT	%

For example, use the following syntax to display the File menu list in Corel Paradox: `sendKeys("%f")`.

The following code moves down 3 menu items: `sendKeys("{down 3}")`.
`Pick the item using the following syntax: sendKeys("~")`.

To combine these three steps into one: `sendKeys("%f{down 3}~")`

To specify that SHIFT, CTRL, and (or) ALT must be held down while one or more keys are pressed, enclose the key codes in parentheses. For example, if SHIFT is pressed while `a` and `b` are pressed, use `"+(ab)"`. If SHIFT is pressed while `a` is pressed, and `b` is pressed without SHIFT, use `"+ab"`.

To specify repeating keys, enclose a string and a number in braces {*key number*}. For example, {`left 42`} specifies you must press the left arrow key 42 times; and {`h 9`} means you must press h 9 times.

Special commands

The following are special commands you can include as part of the *keyText* argument:

- **{delay value}**
  delay sets the delay (in milliseconds) between keystrokes. `{delay 1000} wait`s 1 second between keystrokes; this is approximate and may vary if SHIFT, Alt, or CTRL are set. If the actual time to execute the command is longer, you may see additional delays.

  delay is mainly used to let dialog boxes display. Without it the keys are sent at full speed, and Windows processes the keys too quickly to paint the dialog box on the screen. **delay** remains in effect until another **delay** or a **sendKeys** statement executes; it does not affect action commands.
- **{action integervalue}**
  action sends an action to the object in the form that issued the sendKeys statement. It allows you to gain control while sendKeys executes, to inspect the state of forms or dialog boxes. *integervalue* is a value between 0 and 2047. Do not call any methods or procedures that wait for user input, and do not open a form or report.
- **{cmt comment}**
  cmt lets you insert comments. *comment* represents your remarks; all characters are allowed.
- **{beginexact}text {endexact}**
  sendKeys normally ignores carriage returns and line feeds, and assigns meanings to certain characters. To

bypass this processing, enclose the text with `{beginexact}` and `{endexact}`. Once a `{beginexact}` is encountered, all text is processed exactly as is until the `{endexact}`.

If you call **sendKeys** while another **sendKeys** statement is executing, Corel Paradox adds the new key sequence to the end of the event queue.

- **{menu integervalue}**
  This sends a menu command to the active object. `integervalue` represents a value from the menu command constants.

The wait argument

wait specifies whether to wait after keys are sent, or to continue ObjectPAL execution. The recommended setting is False. Windows sometimes stops responding to **sendKeys** if the **wait** parameter is True (e.g., when keys are sent to nested dialogs). Set **wait** to False when changing the working directory or the private directory.

Note

- sendKeys statements are not portable across language barriers.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYSENDKEYSACTIONID;OPAL_TYPE_TIMEREVENT;',0,"Defaultoverview",)} **Related Topics**

sendKeys example

The following example uses the **execute** system procedure to run the Windows Notepad application and then **sendKeys** sends keystrokes to Notepad twice and saves the file as TWOLINES.TXT:

```
method pushButton(var eventInfo event)

   execute("notepad.exe")          ; run Notepad.
   sleep(1000)
   ; write a short note.

   sendKeys("this is the first line of a 2-line note.~")
   sendKeys("this is the second line of a 2-line note.")

   ; send alt+f, s to choose File, Save.

   sendKeys("%fs")

   ; send a filename to the dialog box, and
   ; send enter to save the file.

 sendKeys("twolines.txt~")

   ; send Alt+f4 to close Notepad.

   sendKeys("%{f4}")

endMethod
```

sendKeysActionID method

Allows the **sendKeys** procedure to notify you when the **sendKeys** queue is empty.

Syntax

```
sendKeysActionID ( const id SmallInt )
```

Description

sendKeysActionID allows the **sendKeys** procedure to notify you when the **sendKeys** queue is empty. The argument *id* is a user-defined action constant whose value is between the IdRanges constants UserAction and UserActionMax. *id* is sent to the form's active object (or to the form itself if there is no active object) that issued the **sendKeys** method.

The code used to process **sendKeysActionID** is usually placed at the form level. If there is an active object, it receives the ID in its **action** method. The default, however, is to bubble the action ID to the form.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;',0,"Defaultoverview",)} Related Topics

sendKeysActionID example

The following example specifies the action ID value sent when the queue is empty. Suppose a form contains an unbound field and a button. The following code is attached to the form's Const window:

```
const
   kMyCustomAction = 1
endConst
```

The following code is attached to the form's built-in **action** method.

```
method action(var eventInfo ActionEvent)
  if eventInfo.id() = UserAction + kMyCustomAction then
     message("sendKeys has finished sending")
  endIf
endMethod
```

The following code is attached to a button's built-in **pushButton** method.

```
method pushButton(var eventInfo Event)
   ; Send keys but do not wait.
   sendKeys("This is some text", FALSE)


   ; Set the action id to send when the queue is empty.
   sendKeysActionID(UserAction + kMyCustomAction)
endMethod
```

setDefaultPrinterStyleSheet procedure

Specifies a default printer style sheet.

Syntax

```
setDefaultPrinterStyleSheet ( const fileName String )
```

Description

setDefaultPrinterStyleSheet sets the Corel Paradox style sheet, specified by *fileName*, as the default for documents designed for the printer. If *fileName* does not specify a full path, **setDefaultPrinterStyleSheet** searches the working directory.

Any UIObjects created in forms and reports while the style sheet is active are given the properties and methods of the corresponding prototype objects in the style sheet.

This procedure does not change the properties or methods of existing UIObjects and has no effect on UIObjects in forms and reports that use different style sheets.

Use **getStyleSheet** and **setStyleSheet** to work with style sheets for specific forms and reports.

Use **setDefaultScreenStyleSheet** to specify the name of the default screen style sheet. The screen style sheet is used whenever you create design documents that are designed for the screen.

Note

- Printer style sheet files have an .FP extension and screen style sheet files have and .FT the extension. Printer and screen style sheets are not interchangeable.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYGETDEFAULTPRINTERSTYLESHEET;OPAL_METH_SYGETDEFAULTSCREENSTYLESHEET;OPAL_METH_SYSETDEFAULTSCREENSTYLESHEET;OPAL_METH_FOSAVESTYLESHEET;OPAL_METH_FOGETPROTOPROPERTY;OPAL_METH_FOSETPROTOPROPERTY;',0,"Defaultoverview",)} Related Topics

setDefaultPrinterStyleSheet example

The following example calls **getDefaultPrinterStyleSheet** to determine the current default style sheet. If the style sheet is not COREL.FT, the code calls **setDefaultPrinterStyleSheet** to set it. The code then calls **getDefaultPrinterStyleSheet** again to make sure it was reset successfully.

setDefaultPrinterStyleSheet requires double backslashes in the path, but **getDefaultPrinterStyleSheet** returns single backslashes.

```
method pushButton(var eventInfo Event)
   ; Get and set the system style sheet.
   if getDefaultPrinterStyleSheet() <> "c:\\Corel\\Suite8\\Paradox\\Corel.fp" then
      setDefaultPrinterStyleSheet("c:\\Corel\\Suite8\\Paradox\\Corel.fp")
      if getDefaultPrinterStyleSheet() <> "c:\\Corel\\Suite8\\Paradox\\Corel.fp" then
         msgStop("Problem", "Could not set the style sheet.")
      endIf
   endIf
endMethod
```

setDefaultScreenStyleSheet procedure

Specifies a default screen style sheet.

Syntax

```
setDefaultScreenStyleSheet ( const fileName String )
```

Description

setDefaultScreenStyleSheet sets the Corel Paradox style sheet specified by *fileName* as the default for documents designed for the screen. If *fileName* does not specify a full path, **setDefaultScreenStyleSheet** searches the working directory.

Any UIObjects created in forms and reports while the style sheet is active are given the properties and methods of the corresponding prototype objects in the style sheet.

This procedure does not change the properties or methods of existing UIObjects and has no effect on UIObjects in forms and reports that use different style sheets.

Use **getStyleSheet** and **setStyleSheet** to work with style sheets for specific forms and reports.

Use setDefaultScreenStyleSheet to specify the name of the default screen style sheet. The screen style sheet is used whenever you create design documents that are designed for the screen.

Note

- Printer style sheet files have an .FP extension and screen style sheet files have and .FT the extension. Printer and screen style sheets are not interchangeable.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYGETDEFAULTPRINTERSTYLESHEET;OPAL_METH_SYGETDEFAULTSCREENSTYLESHEET;OPAL_METH_FOSAVESTYLESHEET;OPAL_METH_FOGETPROTOPROPERTY;OPAL_METH_FOSETPROTOPROPERTY;',0,"Defaultoverview",)} Related Topics

setDefaultScreenStyleSheet example

The following example calls **getDefaultScreenStyleSheet** to determine the current system style sheet. If it is not COREL.FT, **setDefaultScreenStyleSheet** sets it. The code then makes sure it was set successfully.

```
method pushButton(var eventInfo Event)
   ; Get and set the system style sheet.
   if getDefaultScreenStyleSheet() <> "c:\\Corel\\Suite8\\Paradox\\Corel.ft" then
      setDefaultScreenStyleSheet("c:\\Corel\\Suite8\\Paradox\\Corel.ft")
      if getDefaultScreenStyleSheet() <> "c:\\Corel\\Suite8\\Paradox\\Corel.ft" then
         msgStop("Problem", "Could not set the style sheet.")
      endIf
   endIf
endMethod
```

setDesktopPreference procedure

Sets a desktop preference.

Syntax

```
setDesktopPreference ( const section AnyType, const name AnyType, const value AnyType ) Logical
```

Description

setDesktopPreference sets the desktop preference specified by the *section* and *name* arguments. The *value* argument corresponds to one of the DesktopPreferenceTypes Constants.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYGETDESKTOPPREFERENCE;',0,"Defaultoverview",)}

Related Topics

setDesktopPreference example

The following example sets the title name preference, retrieves the name, and displays it:

```
method pushButton(var eventInfo Event)
setDesktopPreference( PrefProjectSection, prefTitleName,"Corel Paradox pour Windows" )

x = getDesktopPreference( PrefProjectSection, prefTitleName )

x.view()
endmethod
```

setMouseScreenPosition procedure

Displays the pointer at a specified position.

Syntax

```
1. setMouseScreenPosition ( const mousePosition Point )
2. setMouseScreenPosition ( const x LongInt, const y LongInt )
```

Description

setMouseScreenPosition displays the pointer at the specified position. In Syntax 1, the pointer is displayed at the point specified in *mousePosition.* In Syntax 2 the pointer is displayed at the coordinates specified in twips by *x* and *y*.

Use Point type methods such as **x** and **y** to retrieve more information.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYGMOUS;OPAL_METH_MEGPOS;OPAL_METH_MESPOS;',0,"Defaultoverview",)} Related Topics

setMouseScreenPosition example

See the **getMouseScreenPosition** example.

setMouseShape procedure

Sets the shape of the pointer.

Syntax

```
setMouseShape ( const mouseShapeId LongInt [,const persist Logical] ) LongInt
```

Description

setMouseShape sets the shape of the pointer. The argument *mouseShapeId* specifies the shape of the pointer. ObjectPAL provides MouseShapes constants for this purpose.

If **persist** is true then the pointer will be persistent (will not change shape) to objects that implicitly change the shape of the mouse (e.g., button objects and field objects). **persist** will not affect where the ObjectPAL developer has explicitly changed the shape of the mouse. For example, in a **mouseEnter** method of an object, **setMouseShape** will override mouse persistence. **persist** does not affect ActiveX or native Windows controls.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYGMOUS;OPAL_METH_SYISMOUSEPERSISTENT;OPAL_METH_SYSMPOS;OPAL_METH_SYSETMOUSESHAPEFROMFILE;',0,"Defaultoverview",)} Related Topics

setMouseShape example

In the following example, a form has two buttons: *btnNonPersistent* and *btnPersistent*. The **pushButton** method of each button uses **setMouseShape** to set the mouse shape of the cursor; the first with persistence set to False, the second with persistence set to True. The second button, *btnPersistent* also contains a **mouseEnter** method which will use **isMousePersistent** to evaluate the persistency of the pointer and return it to its original state.

When the first button is pressed, the pointer changes. However, when the pointer moves off the button, the pointer returns to its original setting. When the second button is pressed, the pointer changes and remains that way until the pointer moves back over the second button. This triggers the mouseEnter method of the second button and return the pointer back to its original state.

The following code is attached to the **pushButton** method for *btnNonPersistent*:

```
; btnNonPersistent::pushButton
method pushButton(var eventInfo Event)
   ;// Set the shape to international symbol for No - non-persistent
   setMouseShape(MouseNo,FALSE)
endMethod
```

The following code is attached to the **pushButton** method for *btnPersistent*:

```
; btnPersistent::pushButton
method pushButton(var eventInfo Event)
   ;// Set the shape to international symbol for No - persistent
   setMouseShape(MouseNo,TRUE)
endMethod
```

The following code is attached to the **mouseEnter** method for *btnPersistent*:

```
; btnPersistent::mouseEnter
method mouseEnter(var eventInfo MouseEvent)
   if isMousePersistent() then
      ;// If its persistent, set it back to the arrow cursor
      setMouseShape(MouseArrow,FALSE)
   endIf
endMethod
```

setMouseShapeFromFile method

Specifies the shape of the pointer.

Syntax

```
setMouseShapeFromFile ( const fileName String [,const persist Logical] ) LongInt
```

Description

setMouseShapeFromFile specifies the shape of the pointer based on data contained in *fileName*. *fileName* is a *.CUR or *.ANI file that supports paths and aliases. If *fileName* does not exist, a warning is generated. setMouseShapeFromFile returns a LongInt handle to the mouse shape.

If *persist* is True then the pointer is persistent (will not change shape) to objects that implicitly change the shape of the mouse (e.g., button objects and field objects). *persist* does not affect where the ObjectPAL developer has explicitly changed the shape of the mouse. For example, in a **mouseEnter** method of an object, **setMouseShape** overrides mouse persistence. *persist* does not affect ActiveX or native Windows controls.

Example

setMouseShapeFromFile example

In the following example, a form has two buttons: *btnNonPersistent* and *btnPersistent*. The **pushButton** method of each button uses **setMouseShapeFromFile** to set the mouse shape of the cursor to an animated cursor provided with Windows 95 and Windows NT; the first with persistence set to false, the second with persistence set to true. The second button, *btnPersistent* also contains a **mouseEnter** method which will use **isMousePersistent** to evaluate the persistency of the pointer and return it to its original state.

When the first button is pressed, the pointer changes. However, when the pointer moves off the button, the pointer returns to its original setting. When the second button is pressed, the pointer changes and remains that way until the pointer is moved back over the second button. This triggers the **mouseEnter** method of the second button and returns the pointer back to its original state. Each **pushButton** method will determine which operating system its running under to determine where to find the animated cursor file.

The following code is attached to the **pushButton** method for *btnNonPersistent*:

```
; btnNonPersistent::pushButton
method pushButton(var eventInfo Event)
var
   sysDyn       DynArray[] AnyType
   mouseHandle  LongInt
endVar
   sysInfo(sysDyn)
   if sysDyn["WindowsPlatform"] = "WIN95" then
                  ;// if Windows 95
      mouseHandle = setMouseShapeFromFile( windowsDir() +
                  "\\CURSORS\\HOURGLAS.ANI", FALSE)
   else
         ;// if Windows NT
      mouseHandle = setMouseShapeFromFile( windowsSystemDir() +
                  "\\HOURGLAS.ANI", FALSE)
   endIf
endMethod
```

The following code is attached to the **pushButton** method for *btnPersistent*:

```
; btnPersistent::pushButton
method pushButton(var eventInfo Event)
var
   sysDyn       DynArray[] AnyType
   mouseHandle  LongInt
endVar
   sysInfo(sysDyn)
   if sysDyn["WindowsPlatform"] = "WIN95" then
         ;// if Windows 95
      mouseHandle = setMouseShapeFromFile( windowsDir() +
                  "\\CURSORS\\HOURGLAS.ANI", TRUE)
   else
         ;// if Windows NT
      mouseHandle = setMouseShapeFromFile( windowsSystemDir() +
                  "\\HOURGLAS.ANI", TRUE)
   endIf
endMethod
```

The following code is attached to the **mouseEnter** method for *btnPersistent*:

```
; btnPersistent::mouseEnter
method mouseEnterpushButton(var eventInfo MouseEvent)
   if isMousePersistent() then
         ;// If its persistent, set it back to the arrow cursor
      setMouseShap(MouseArrow,FALSE)
   endIf
endMethod
```

setRegistryValue method

Sets a value in the registry.

Syntax

```
setRegistryValue ( const key String, const value String, const data AnyType, const rootKey LongInt ) Logical
```

Description

setRegistryValue writes data to a specified value of a registry key. If the *key* or *value* do not exist, then they will be created. If *data* is empty then only *key* is created. If *value* is empty, then *key* and *data* are created.

key is a path similar to a file path. However, wildcards are not expanded in the *key*. *key* cannot contain a single backslash and cannot be empty. Its size is limited to 65,534 bytes.

The *value* is a string that is limited to 65,534 bytes. *value* can contain backslashes and can be empty. **setRegistryValue** returns True if successful; otherwise, it returns False.

data accepts the following types:

ObjectPAL Type	Registry type	Size limitation
Currency	String	32k
Date	String	32k
DateTime	String	32k
Logical	String	32k
LongInt	DWORD	32k
Memo	String	32k
Number	String	32k
Point	String	32k
SmallInt	DWORD	32k
String	String	32k
Time	String	32k

rootKey is analogous to a directory drive. Set *rootKey* with the predefined RegistryKeyType Constants.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYDELETEREGISTRYKEY;OPAL_METH_SYENUMREGISTRYKEYS;OPAL_METH_SYENUMREGISTRYVALUENAMES;OPAL_METH_SYGETREGISTRYVALUE;OPAL_METH_SYSEARCHREGISTRY;',0,"Defaultoverview",)} **Related Topics**

setRegistryValue example

The following example sets the current ObjectPAL level in the registry:

```
var
   strLevel   String
endvar

;// create key, value and data in regCurrentUser
setRegistryValue( "Software\\Corel\\Myapp\\Settings", "ObjectValue", "An object",
regKeyCurrentUser )
```

setUserLevel procedure

Sets your ObjectPAL level (Beginner or Advanced). Beginner restricts the methods displayed for each object in the Integrated Development Environment (IDE) to those a new ObjectPAL user would likely need; Advanced displays all methods.

Syntax

```
setUserLevel ( const level String )
```

Description

setUserLevel Sets your ObjectPAL level (Beginner or Advanced). Use **getUserLevel** to return the current setting.

Notes

- The ObjectPAL level setting *does not* affect how code executes; it only affects what is displayed in the user interface.
- The advanced setting is highly recommended.

Example

**{button ,AL(`OPAL_TYPE_SYSTEM;INTRO_INTRO;OPAL_METH_SYSETUSERLEVEL;;;',0,"Defaultoverview",)
} Related Topics**

setUserLevel example

Uses **getUserLevel** to determine if the ObjectPAL user level is set to Beginner. If the ObjectPAL level is set to Beginner, **setUserLevel** sets it to Advanced. If the ObjectPAL user level is already set to Advanced, the code sends a message stating this to the Status Bar.

```
;setToAdvanced::pushButton
method pushButton(var eventInfo Event)
   if getUserLevel() = "Beginner" then
      setUserLevel("Advanced")
      message("ObjectPAL level is now set to Advanced")
   else
      message("ObjectPAL level was already set to Advanced")
   endIf
endmethod
```

sleep procedure

The following example produces a delay of a specified duration.

Syntax

```
sleep ( [ const numberOfMilliSeconds LongInt ] )
```

Description

sleep disables the executing form for the number of milliseconds specified in *numberofMilliseconds*. **sleep** does not disable the desktop or stop timer events. When the form is disabled, it cannot receive keystrokes, mouse events or focus.

Note

- When **sleep** is called with no argument, it does *not* disable the form. Instead, it causes the current method to yield to Windows to let a single pending message be processed.

Example

**{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_TYPE_TIMEREVENT;OPAL_METH_FOWAIT;',0,"Defaultoverview",)
} Related Topics**

sleep example

The following example displays a message in the status line and then waits five seconds before displaying a second message:

```
; goToSleep::pushButton
method pushButton(var eventInfo Event)
var
   yourTurn SmallInt
endVar
yourTurn = 5000
beep()
message("Next message in 5 seconds.")
sleep(yourTurn)                     ; waits for 5 seconds
message("5 seconds have elapsed.")
endMethod
```

sound procedure

Creates a sound of specified frequency and duration.

Syntax

sound (const ***freqHertz***, const ***durationMilliSecs*** LongInt **)**

Description

sound creates a sound of the frequency specified by *freqHertz* (in Hertz) for a time *durationMilliSecs* (in milliseconds) . Frequency values can range from 1 to 50,000 Hertz. The sound is played through the computer's internal speaker, and not the system sound card.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYBEEP;',0,"Defaultoverview",)} **Related Topics**

sound example

The following example uses the **pushButton** method for *makeMusic* to declare constants for frequency values in a scale. These notes specify the *frequency* argument in the calls to the **sound** method. After playing a few bars from a tune, the method demonstrates the calculation for notes in a chromatic scale (proceeds by half notes).

```
; makeMusic::pushButton
method pushButton(var eventInfo Event)
var
   quarterNote, octave, note LongInt
   power                     Number
endVar
; frequency values for notes in a scale
const
  noteA1  = 110
  noteA#1 = 116
  noteB1  = 123
  noteC1  = 130
  noteC#1 = 138
  noteD1  = 146
  noteD#1 = 155
  noteE1  = 164
  noteF1  = 174
  noteF#1 = 184
  noteG1  = 195
  noteG#1 = 207
  noteA2  = 220
  noteA#2 = 234
  noteB2  = 249
  noteC2  = 265
  noteC#2 = 282
  noteD2  = 300
endConst
; several bars from Peter and the Wolf
sound(noteA1, 200)
sound(noteD1, 150)
sound(noteF#1, 50)
sound(noteA2, 100)
sound(noteB2, 100)
sound(noteA2, 150)


sound(noteF#1, 50)
sound(noteA2, 100)
sound(noteB2, 100)
sound(noteC#2, 150)
sound(noteD2, 50)
sound(noteA2, 100)
sound(noteF#1, 100)
sound(noteD1, 100)
sleep(1000)


; play a few chromatic scales
quarterNote = 120
for octave from 0 to 1
   for note from 0 to 11
      sound(int(pow(2, octave + note / 12.0) * 110), quarterNote)
   endFor
endFor
sound(int(pow(2, 2) * 110), quarterNote) ; finish out the scale
endMethod
```

sysInfo procedure

Creates a dynamic array of information about the system running Corel Paradox.

Syntax

```
sysInfo ( var info DynArray[ ] AnyType )
```

Description

sysInfo creates a dynamic array of information about the system running Corel Paradox. Declare a dynamic array named *info* before calling **sysInfo** . *info* contains indexes for system attributes and their values. The following table describes the structure of *info*:

System Attribute Index	Definition
AnsiCodePage	The ANSI (Windows) code page loaded by Windows
AreMouseButtonsSwapped	Functions of the left and right mouse buttons are reversed
CodePage	The code page currently loaded by Windows
CPU	Processor type
Edition	Corel Paradox edition (e.g., Standard)
EngineDate	Creation date of database engine
EngineLanguageID	The language used for Borland Database Engine (BDE) messages and QUERY BY EXAMPLE (QBE) keywords, shown in the list of language identifiers
EngineVersion	Version number of database engine
FullHeight	Vertical working area in a maximized window (in pixels)
FullWidth	Horizontal working area in a maximized window (in pixels)
IconHeight	Height of icons (in pixels)
IconWidth	Width of icons (in pixels)
KeyboardFNKeys	Number of function keys
KeyboardLayoutID	The layout name for the currently loaded keyboard (usually a language ID)
KeyboardSubType	An OEM-dependent value
KeyboardType	Keyboard type and manufacturer
LanguageDriver	Default language drivers for Corel Paradox tables
LocalShare	Reports whether Local Share is active
Memory	Available memory in bytes, including swap file (if present)
Mouse	The number of mouses attached to the system
NetDir	The path to PDOXUSRS.NET
NetProtocol	Network protocol
NetShare	Reports whether Net Share is active
NetType	Network type
ParadoxSystemDir	The path of the Corel Paradox folder
ScreenHeight	Total height of screen (in pixels)
ScreenWidth	Total width of screen (in pixels)
StartupDir	The full path (including the drive ID letter) to your start-up folder (the folder from which Corel Paradox was launched)
SystemDefaultLCID	The system default locale ID (a 32-bit value which is the combination of a language ID and a sort ID)
UserDefaultLCID	The user default locale ID
UserName	Network user name
WindowsBuild#	The internal build number
WindowsDir	Path to the WINDOWS directory (folder)
WindowsPlatform	Win95, NT, or WIN32s
WindowsSystemDir	Path to the WINDOWS\SYSTEM directory (folder)
WindowsText	Arbitrary information

WindowsVersion	Windows version number

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYRPROF;OPAL_METH_SYRENV;OPAL_METH_SYRESOURCEINFO;OPAL_METH_SYVERSION;',0,"Defaultoverview",)} **Related Topics**

sysInfo example

The following example writes system information to a dynamic array named *userSys* and then displays *userSys* in a View dialog box:

```
; showSysInfo::pushButton
method pushButton(var eventInfo Event)
var
   userSys DynArray[] AnyType
endVar
sysInfo(userSys)   ; fill the array with system information
userSys.view()     ; show the array
endMethod
```

Language identifiers

Language identifiers consists of the primary language ID and the sub_language ID.

The following codes are included in the primary language IDs:

Code	Language	Code	Language
0x0401	Arabic	0x0415	Polish
0x0402	Bulgarian	0x0416	Brazilian Portuguese
0x0403	Catalan	0x0417	Rhaeto-Romanic
0x0404	Traditional Chinese	0x0418	Romanian
0x0405	Czech	0x0419	Russian
0x0406	Danish	0x041A	Croato-Serbian (Latin)
0x0407	German	0x041B	Slovak
0x0408	Greek	0x041C	Albanian
0x0409	U.S. English	0x041D	Swedish
0x040A	Castilian Spanish	0x041E	Thai
0x040B	Finnish	0x041F	Turkish
0x040C	French	0x0420	Urdu
0x040D	Hebrew	0x0421	Bahasa
0x040E	Hungarian	0x0804	Simplified Chinese
0x040F	Icelandic	0x0807	Swiss German
0x0410	Italian	0x0809	U.K. English
0x0411	Japanese	0x080A	Mexican Spanish
0x0412	Korean	0x080C	Belgian French
0x0413	Dutch	0x0C0C	Canadian French
0x0414	Norwegian - Bokml	0x100C	Swiss French
0x0810	Swiss Italian	0x0816	Portuguese
0x0813	Belgian Dutch	0x081A	Serbo-Croatian(Cyrillic)
0x0814	Norwegian - Nynorsk		

tracerClear procedure

Clears the Tracer window.

Syntax

```
tracerClear ( )
```

Description

tracerClear clears the Tracer window. You can open the Tracer window with the **tracerOn** procedure at run time, or by clicking View, Tracer in the ObjectPAL Editor.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYTRACERHIDE;OPAL_METH_SYTRACEROFF;OPAL_METH_SYTRACERON;OPAL_METH_SYTRACERSAVE;OPAL_METH_SYTRACERSHOW;OPAL_METH_SYTRACERTOTOP;OPAL_METH_SYTRACERWRITE;',0,"Defaultoverview",)} Related Topics

tracerClear example

The following example clears the Tracer window. Assume that the Tracer window is open and contains information.

```
; wipeTracer::pushButton
method pushButton(var eventInfo Event)
tracerClear()                     ; clear the Tracer window
endMethod
```

tracerHide procedure

Hides the Tracer window.

Syntax

```
tracerHide ( )
```

Description

tracerHide hides the Tracer window. This procedure makes the Tracer window invisible but does not clear or close it. To view the Tracer again, use **tracerShow**.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYTRACERCLEAR;OPAL_METH_SYTRACEROFF;OPAL_METH_SYTRACERON;OPAL_METH_SYTRACERSAVE;OPAL_METH_SYTRACERSHOW;OPAL_METH_SYTRACERTOTOP;OPAL_METH_SYTRACERWRITE;',0,"Defaultoverview",)} Related Topics

tracerHide example

The following example hides the Tracer window, pauses and then displays it again. Assume that the Tracer window is open.

```
; toggleTracerWin::pushButton
method pushButton(var eventInfo Event)
tracerHide()                      ; make the Tracer window invisible
message("Hiding Tracer window. Pausing...")
sleep(2000)
message("Showing Tracer window.")
tracerShow()                      ; make the Tracer window visible again
tracerToTop()                     ; bring it to the top
endMethod
```

tracerOff procedure

Closes the Tracer window.

Syntax

```
tracerOff ( )
```

Description

tracerOff closes the Tracer window. This procedure stops writing code traces to the Tracer window. You can resume tracing code with the **tracerOn** procedure. By default, tracing is turned on when the Tracer window is opened.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYTRACERCLEAR;OPAL_METH_SYTRACERHIDE;OPAL_METH_SYTRACERON;OPAL_METH_SYTRACERSAVE;OPAL_METH_SYTRACERSHOW;OPAL_METH_SYTRACERTOTOP;OPAL_METH_SYTRACERWRITE;',0,"Defaultoverview",)} Related Topics

tracerOff example

The following example turns off code tracing:

```
; stopTracer::pushButton
method pushButton(var eventInfo Event)
tracerOff()                     ; close the Tracer window
endMethod
```

tracerOn procedure

Activates code tracing.

Syntax

```
tracerOn ( )
```

Description

tracerOn activates code tracing. This procedure resumes writing code traces to the Tracer window.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYTRACERCLEAR;OPAL_METH_SYTRACERHIDE;OPAL_METH_SYTRACEROFF;OPAL_METH_SYTRACERSAVE;OPAL_METH_SYTRACERSHOW;OPAL_METH_SYTRACERTOTOP;OPAL_METH_SYTRACERWRITE;',0,"Defaultoverview",)} Related Topics

tracerOn example

The following example reactivates code tracing:

```
; startTracer::pushButton
method pushButton(var eventInfo Event)
tracerOn()                      ; reactivate the Tracer window
endMethod
```

tracerSave procedure

Saves the contents of the Tracer window to a file.

Syntax

```
tracerSave ( const fileName String )
```

Description

tracerSave saves the contents of the Tracer window to the file specified by *fileName*.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYTRACERCLEAR;OPAL_METH_SYTRACERHIDE;OPAL_METH_SYTRACEROFF;OPAL_METH_SYTRACERON;OPAL_METH_SYTRACERSHOW;OPAL_METH_SYTRACERTOTOP;OPAL_METH_SYTRACERWRITE;',0,"Defaultoverview",)} **Related Topics**

tracerSave example

The following example saves the contents of the Tracer window to a file named MYTRACE.TXT:

```
; saveTracerToFile::pushButton
method pushButton(var eventInfo Event)
tracerSave("mytrace.txt")         ; save the Tracer window to a file
endMethod
```

tracerShow procedure

Makes the Tracer window visible.

Syntax

```
tracerShow ( )
```

Description

tracerShow makes the Tracer window visible. You can make the Tracer window invisible using the **tracerHide** procedure.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYTRACERCLEAR;OPAL_METH_SYTRACERHIDE;OPAL_METH_SYTRACEROFF;OPAL_METH_SYTRACERON;OPAL_METH_SYTRACERSAVE;OPAL_METH_SYTRACERTOTOP;OPAL_METH_SYTRACERWRITE;',0,"Defaultoverview",)} Related Topics

tracerShow example

See the **tracerHide** example.

tracerToTop procedure

Positions the Tracer window on top of all other windows on the desktop.

Syntax

```
tracerToTop ( )
```

Description

tracerToTop places the Tracer window on top of all other windows on the desktop.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYTRACERCLEAR;OPAL_METH_SYTRACERHIDE;OPAL_METH_SYTRACEROFF;OPAL_METH_SYTRACERON;OPAL_METH_SYTRACERSAVE;OPAL_METH_SYTRACERSHOW;OPAL_METH_SYTRACERWRITE;',0,"Defaultoverview",)} **Related Topics**

tracerToTop example

See the **tracerWrite** example.

tracerWrite procedure

Writes a message to the Tracer window.

Syntax

```
tracerWrite ( const message String [ , const message String ] * )
```

Description

tracerWrite writes a message to the Tracer window.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYTRACERCLEAR;OPAL_METH_SYTRACERHIDE;OPAL_METH_SYTRACEROFF;OPAL_METH_SYTRACERON;OPAL_METH_SYTRACERSAVE;OPAL_METH_SYTRACERSHOW;OPAL_METH_SYTRACERTOTOP;',0,"Defaultoverview",)} Related Topics

tracerWrite example

The following example logs a message to the Tracer window and places the Tracer window on top of all other windows on the desktop:

```
; logTracerMsg::pushButton
method pushButton(var eventInfo Event)
tracerWrite("Tracer hit by " + String(self.Name) +
            " at " + String(time()))              ; log a message
tracerToTop()            ; make the Tracer window the top-layer window
endMethod
```

twipsToPixels procedure

Converts screen coordinates from twips to pixels.

Syntax

twipsToPixels (const ***twips*** Point) Point

Description

twipstToPixels converts the screen coordinates specified in *twips* from twips to pixels.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYPIXELSTOTWIPS;',0,"Defaultoverview",)} Related Topics

twipsToPixels example

See the **pixelsToTwips** example.

version procedure

Returns the Corel Paradox version number.

Syntax

```
version ( ) String
```

Description

version returns the Corel Paradox version number. If you have more than one version installed, **version** returns the version number of the active application.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYSYSINFO;',0,"Defaultoverview",)} Related Topics

version example

The following example uses the **pushButton** method for *showVersion* to show which version of Corel Paradox is active:

```
; showVersion::pushButton
method pushButton(var eventInfo Event)
   msgInfo("FYI", "You are running version " + version() + ".")
endMethod
```

winGetMessageID procedure

Returns the ID of a Windows message.

Syntax

```
winGetMessageID ( const msgName String ) SmallInt
```

Description

winGetMessageID returns the integer value of the Windows message represented by the string specified in *msgName*. Messages may include WM_CLOSE (sent as a signal that a window or application should terminate), and WM_ACTIVATE (sent when a window is activated or deactivated).

winGetMessageID returns 0 if *msgName* is not recognized as a Windows message. For more information, see your Windows programming documentation.

Note

- **winGetMessageID** should only be used by Windows programmers who are familiar with Windows messages.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYWINPOSTMESSAGE;OPAL_METH_SYWINSENDMESSAGE;OPAL_METH_MNDATA;OPAL_METH_MNID;OPAL_METH_FOWINDOWCLIENTHANDLE;OPAL_METH_FOWINDOWHANDLE;',0,"Defaultoverview",)} Related Topics

winGetMessageID example

The following example displays the integer value of the Windows message WM_LBUTTONDOWN:

```
method pushButton(var eventInfo event)
  var
    smMsgID   SmallInt
    stMsgName   String
  endVar


  stMsgName = "WM_LBUTTONDOWN"
  smMsgID = winGetMessageID(stMsgName)
  smMsgID.view(stMsgName) ; Displays 513 in Win32.
  ; The value may be different in other versions of Windows.
endMethod
```

winPostMessage procedure

Posts a message to Windows.

Syntax

```
winPostMessage ( const hWnd LongInt, const msg LongInt, const wParam LongInt, const lParam LongInt ) Logical
```

Description

winPostMessage posts a message to Windows. Unlike **winSendMessage**, which dispatches its message immediately, **winPostMessage** method adds its message to the end of the Windows message queue. Messages in the queue are dispatched in the order than they appear. Windows determines which arguments are valid to **winPostMessage** For more information, see your Windows programming documentation.

Note

- **winPostMessage** should only be used by Windows programmers who are familiar with Windows messages.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYWINGETMESSAGEID;OPAL_METH_SYWINSENDMESSAGE;OPAL_METH_MNDATA;OPAL_METH_MNID;OPAL_METH_FOWINDOWCLIENTHANDLE;OPAL_METH_FOWINDOWHANDLE;',0,"Defaultoverview",)} Related Topics

winPostMessage example

See the **winSendMessage** example.

winSendMessage procedure

Sends a message to Windows.

Syntax

```
winSendMessage ( const hWnd LongInt, const msg LongInt, const  wParam LongInt, const lParam LongInt ) LongInt
```

Description

winSendMessage sends a message to Windows. Windows determines which arguments are valid to **winSendMessage** For more information, see your Windows programming documentation.

Note

- **winPostMessage** should only be used by Windows programmers who are familiar with Windows messages.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYWINGETMESSAGEID;OPAL_METH_SYWINPOSTMESSAGE;OPAL_METH_MNDATA;OPAL_METH_MNID;OPAL_METH_FOWINDOWCLIENTHANDLE;OPAL_METH_FOWINDOWHANDLE;',0,"Defaultoverview",)} Related Topics

winSendMessage example

The following example opens Notepad and calls **enumWindowNames** to create a table of data about the windows currently open on your system. The code then searches the table for information about Notepad, and gets the handle for that window. Next, calls **winGetMessageID** to retrieve the integer value of the command represented by the string "WM_CLOSE." Finally, the code calls **winSendMessage** with the window handle and command value as arguments. The message is dispatched to Windows, and Notepad is closed. To add the message to the end of the Windows message queue, call **winPostMessage** instead of **winSendMessage**.

```
method pushButton(var eventInfo Event)
  var
    tcOpenWin   TCursor
    tbOpenWin   Table
    stTbName    String
    siWinHandle,
    siWinMsgID  SmallInt
  endVar


  stTbName = ":PRIV:openWin"


  execute("Notepad.exe", No, ExeShowNormal)           ; Run Notepad.
  sleep(1000)                  ;  Pause so you can see what happens.


  enumWindowNames(stTbName)                     ; List open windows.


  tcOpenWin.open(stTbName)
  ; Locate the Notepad window in the list of names.
  if tcOpenWin.locatePattern("ClassName", "Notepad") then
    ; Get the Windows handle for the Notepad window.
    siWinHandle = tcOpenWin."Handle"
    ; Get the Windows message ID for WM_CLOSE to close the window.
    siWinMsgID = winGetMessageId("WM_CLOSE")
    ; Send the specified message to the specified window.
    winSendMessage(siWinHandle, siWinMsgID, 0, 0)
  else
    errorShow()
  endIf
endmethod
```

writeEnvironmentString procedure

Sets a variable in the Corel Paradox copy of the DOS environment.

Syntax

```
writeEnvironmentString ( const key String, const value String ) Logical
```

Description

writeEnvironmentString sets a variable in the Corel Paradox copy of the DOS environment. When Corel Paradox launches, a copy of the DOS environment is made. **writeEnvironmentString** writes to that copy but changes are not written to the DOS environment.

You can use the SET command to assign environment variables. These assigned values control the appearance and function of DOS and some batch files. Some common environment variables include PATH, PROMPT, and COMSPEC. For more information, the SET command in your DOS manuals.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYRENV;OPAL_METH_SYWPROF;',0,"Defaultoverview",)}

Related Topics

writeEnvironmentString example

See the **readEnvironmentString** example.

writeProfileString procedure

Writes system information to a file.

Syntax

```
writeProfileString ( const fileName String, const section String, const key String, const value String ) Logical
```

Description

writeProfileString writes system information to a specified file on your system. If you specify a filename without a path, this method searches for the file in the WINDOWS directory (folder).

Typically, you use this method to modify your WIN.INI file. In this case, *fileName* would be WIN.INI. Sections are defined by square brackets and reside on a separate line in the WIN.INI file. To specify a section, simply type the string or section name (e.g., to specify the [windows] section, type "windows').

In each section, a value marker is followed by an equal sign (e.g., Beep =). The equal sign is not required when you specify the value of *key*.

Example

{button ,AL(`OPAL_TYPE_SYSTEM;OPAL_METH_SYRPROF;OPAL_METH_SYWRITEENVIRONMENTSTRING;OPAL_METH_FSWDIR;',0,"Defaultoverview",)} Related Topics

writeProfileString example

See the **readProfileString** example.

Table type

A Table variable describes a table. It differs from a TCursor which is a pointer to a table's data, and from a table frame or a TableView, which are objects that display the data.

You can use Table variables to add, copy, create, and index tables, to perform column calculations in columns, retrieve information about a table's structure, and more. Some table operations require Corel Paradox to create temporary tables in the private directory.

The **create**, **index**, and **sort** structures are basic language elements (not methods or procedures) that operate on Table variables. Table variables cannot be used to edit records—you must use a TCursor or table frame (UIObject) to modify a record in a table.

Methods for the Table type

add
attach
cAverage
cCount
cMax
cMin
cNpv
compact
copy
create
createIndex
cSamStd
cSamVar
cStd
cSum
cVar
delete
dropGenFilter
dropIndex
empty
enumFieldNames
enumFieldNamesInIndex
enumFieldStruct
enumIndexStruct
enumRefIntStruct
enumSecStruct
familyRights
fieldName
fieldNo
fieldType
getGenFilter
getRange
index
isAssigned
isEmpty
isEncrypted
isShared
isTable
lock

nFields
nKeyFields
nRecords
protect
reIndex
reIndexAll
rename
restructure
setExclusive
setGenFilter
setIndex
setRange
setReadOnly
showDeleted
sort
subtract
tableRights
type
unAttach
unlock
unProtect
usesIndexes

Print related ObjectPAL methods and examples

add method/procedure

Adds data from one table to another table.

Syntax

1. add (const ***destTableName*** String [**,** const ***append*** Logical [**,** const ***update*** Logical]] **)** Logical

2. add (const ***destTableVar*** Table [**,** const ***append*** Logical [**,** const ***update*** Logical]] **)** Logical

Description

add adds data from a table to a target table, which can be specified using a String (*destTableName* in Syntax 1) or a Table variable (*destTableVar* in Syntax 2). If the target table does not exist, this method creates it. The source table and the target table can be any types that have compatible field structures.

When set to True, *append* adds records at the end of a non-indexed target table, or at the appropriate place in an indexed target table. When set to True, *update* compares records in both tables, and where key values match, replaces the data in the target table. When both are set to True, records with matching key values are updated, and others are appended. These arguments are optional, but if you specify *update*, you must also specify *append.* By default, both arguments are True.

```
myTable.add(yourTable, False, True) ; specifies update
myTable.add(yourTable)              ; specifies update and append by

                                    ; default
```

Key violations (including validity check violations) are listed in KEYVIOL.DB in the private directory. If KEYVIOL.DB already exists, **add** overwrites it. If KEYVIOL.DB does not exist, this method creates it.

When tables are keyed, **add** uses the keyed fields to determine which records to update and which to append. If the target table is not keyed and *update* is set to True, **add** fails. If the target table is not keyed, the structure of the entire record in the source table must match the record structure in the target table.

DOS

If you are a DOS PAL programmer, you can use the following procedure to operate on tables by specifying the table name, rather than using a variable.

Syntax

1. add (const ***sourceTableName*** String**,** const ***destTableName*** String [**,** const ***append*** Logical [**,** const ***update*** Logical]] **)** Logical

2. add (const ***sourceTableName*** String**,** const ***destTableVar*** Table [**,** const ***append*** Logical [**,** const ***update*** Logical]] **)** Logical

Example
{button ,AL(`OPAL_TYPE_TABLE;OPAL_METH_TBCOPY;OPAL_METH_TBSUB;',0,"Defaultoverview",)}
Related Topics

add example

The following example uses the **pushButton** method for *updateCust* to run a query from an existing file and add records from the *Answer* table to the *Customer* table:

```
; updateCust::pushButton
method pushButton(var eventInfo Event)
var
  newCust Query
  ansTbl Table
  destTbl String
endVar
destTbl = "Customer.db"


newCust.readFromFile("newCust.qbe")


if newCust.executeQBE() then         ; if the query succeeds
  ansTbl.attach(":PRIV:Answer.db")


  ; attempt to add Answer.db records to Customer.db
  if isTable(destTbl) then
    if NOT ansTbl.add(destTbl) then
      errorShow()
    endIf
  else
    msgStop("Error", "Can't find " + destTbl + ".")
  endIf
else
  errorShow("Query failed.")
endIf


endMethod
```

attach method

Associates a Table variable with a table on disk.

Syntax

```
1. attach ( const tableName String ) Logical
2. attach ( const tableName String, const db Database ) Logical
3. attach ( const tableName String, const tableType String ) Logical
4. attach ( const tableName String, const tableType String, const db Database ) Logical
```

Description

attach associates a Table variable with the table specified in *tableName*. Optional arguments *tableType* and *db* specify a table type (Corel Paradox or dBASE) and a database. If you don't specify *tableType*, ObjectPAL determines the table type from the table name's file extension. If you don't specify *db*, ObjectPAL works in the default database.

This method fails if the value of *tableName* is not valid (e.g., the table name doesn't match the table type, or conflicts with the database name). This method returns True if successful; otherwise, it returns False.

Notes

- **attach** does not verify that *tableName* exists, or is a table. Use the **isTable** method to verify a table's existence.
- To free a Table variable completely, use **unAttach**. To associate the Table variable with another table, just use **attach** again; the **unAttach** happens automatically.

Example

{button ,AL(`OPAL_TYPE_TABLE;OPAL_METH_TBCREA;OPAL_METH_TBUNATT;',0,"Defaultoverview",)}

Related Topics

attach example

In the following example, the *westTable* Table variable is attached to *Orders* so that **cSum** can be used with that Table variable. This example uses **isTable** to determine whether *Orders* exists in the default database before performing a calculation.

```
; getWestTotal::pushButton
method pushButton(var eventInfo Event)
var
  westTable Table
  westTotal Number
endVar


if isTable("Orders.db")      then


   ; attach to Corel Paradox table Orders in the default database
  westTable.attach("Orders", "Corel Paradox")
   ; get total of Total Invoice field and store result in westTotal
  westTotal = westTable.cSum("Total Invoice")
   ; display total invoices
  msgInfo("Total Invoices", westTotal)


else
  msgInfo("Status", "Can't find Orders.db table.")
endIf


endMethod
```

cAverage method/procedure

Returns the average of values in a column of fields.

Syntax

```
1. cAverage ( const fieldName String ) Number
2. cAverage ( const fieldNum SmallInt ) Number
```

Description

cAverage returns the average of values in the column of fields specified by *fieldName* or *fieldNum*. If the column contains empty fields, **cAverage** uses the **blankAsZero** setting for the session. This method respects the limits of restricted views set by **setRange** or **setGenFilter**.

Throughout the retry period **cAverage** attempts to place a write lock on the table. If a lock cannot be placed, the method fails.

DOS

If you are a DOS PAL programmer, you can use this procedure to operate on tables by specifying the table name, rather than using a variable.

Syntax

```
1. cAverage ( const tableName String, const fieldName String ) Number
2. cAverage ( const tableName String, const fieldNum SmallInt ) Number
```

Example

{button ,AL(`OPAL_TYPE_TABLE;OPAL_METH_TBCCNT;OPAL_METH_TBCMAX;OPAL_METH_TBCMIN;OPAL_METH_TBCSUM;OPAL_METH_TBCSTD;',0,"Defaultoverview",)} Related Topics

cAverage example

The following example uses **cAverage** to calculate the average order size in the *Orders* table. This code is attached to the **pushButton** method for the *getAvgSales* button:

```
; getAvgSales::pushButton
method pushButton(var eventInfo Event)
var
  ordTbl   Table
  avgSales Number
endVar


ordTbl.attach("Orders.db")
avgSales = ordTbl.cAverage("Total Invoice") ; store average invoice total
                                            ; in avgSales
msgInfo("Average Order size", avgSales)     ; display avgSales in a dialog


endMethod
```

cCount method/procedure

Returns the number of nonblank values in a table column.

Syntax

```
1. cCount ( const fieldName String ) LongInt
2. cCount ( const fieldNum SmallInt ) LongInt
```

Description

cCount returns the number of values in the column specified by *fieldName* or *fieldNum*. **cCount** works for all field types. If the column contains numeric values **cCount** this method handles blank values as specified in the **blankAsZero** setting for the session. If the field is non-numeric, **cCount** returns the number of nonblank values in the column of fields.

This method respects the limits of restricted views set by **setRange** or **setGenFilter**.

Throughout the retry period **cCount** attempts to place a read lock on the table. If a lock cannot be placed, the method fails.

DOS

If you are a DOS PAL programmers, you can use this procedure to operate on tables by specifying the table name, rather than using a variable.

Syntax

```
1. cCount ( const tableName String, const fieldName String ) Number
2. cCount ( const tableName String, const fieldNum SmallInt ) Number
```

Example

{button ,AL(`OPAL_TYPE_TABLE;OPAL_METH_TBCAVE;OPAL_METH_TBCMAX;OPAL_METH_TBCMIN;OPAL_METH_TBCSTD;OPAL_METH_TBCSUM;',0,"Defaultoverview",)} Related Topics

cCount example

In the following example, the **pushButton** method for *lineItemInfo* uses **cAverage** and **cCount** to perform calculations on the Qty field in LINEITEM.DB. The code attempts to place a write lock on the table so that changes cannot be made to the table between the calls to **cAverage** and **cCount**. If the lock cannot be placed, the operation is aborted.

```
; lineItemInfo::pushButton
method pushButton(var eventInfo Event)
var
  lineTbl Table
  avgQty Number
  numItems LongInt
endVar
if lineTbl.attach("Lineitem.db") then
  if lineTbl.lock("Write") then           ; if write lock succeeds
    avgQty = lineTbl.cAverage("Qty")
    numItems = lineTbl.cCount(4)          ; assumes Qty is field 4
    lineTbl.unLock("Write")               ; unlock the table
    msgInfo("Average quantity",
            String(avgQty, "\nbased on ", numItems, " items."))
  else
    errorShow("Can't lock Lineitem table.")
  endIf
else
  errorShow("Can't attach to Lineitem table.")
endIf


endMethod
```

cMax method/procedure

Returns the maximum value of a table's column.

Syntax

```
1. cMax ( const fieldName String ) Number
2. cMax ( const fieldNum SmallInt ) Number
```

Description

cMax returns the maximum value in the column of fields specified by *fieldName* or *fieldNum*. **cMax** respects the limits of restricted views set by **setRange** or **setGenFilter**. **cMax** handles blank values as specified in the blankAsZero setting for the session.

Throughout the retry period, this method attempts to place a write lock on the table. If a lock cannot be placed, the method fails.

DOS

If you are a DOS PAL programmer, you can use this procedure to operate on tables by specifying the table name, rather than using a variable.

Syntax

```
1. cMax ( const tableName String, const fieldName String ) Number
2. cMax ( const tableName String, const fieldNum SmallInt ) Number
```

Example

{button ,AL(`OPAL_TYPE_TABLE;OPAL_METH_TBCAVE;OPAL_METH_TBCMIN;OPAL_METH_TBCSTD;OPAL_METH_TBCSUM;OPAL_METH_TBCCNT;',0,"Defaultoverview",)} **Related Topics**

cMax example

The following example displays the maximum value in the Total Invoice field of the *Orders* table:

```
; showMaxOrder::pushButton
method pushButton(var eventInfo Event)
var
  orderTbl Table
endVar


if orderTbl.attach("Orders.db") then
  ; display maximum order in a dialog box
  msgInfo("Biggest Order in History", orderTbl.cMax("Total Invoice"))
else
  msgStop("Sorry", "Can't open Orders table.")
endIf


endMethod
```

cMin method/procedure

Returns the minimum value of a table's column.

Syntax

```
1. cMin ( const fieldName String ) Number
2. cMin ( const fieldNum SmallInt ) Number
```

Description

cMin returns the minimum value in the column of fields specified by *fieldName* or *fieldNum*. This method respects the limits of restricted views set by **setRange** or **setGenFilter**. **cMin** handles blank values as specified in the **blankAsZero** setting for the session.

Throughout the retry period, this method attempts to place a read lock on the table. If a lock cannot be placed, the method fails.

DOS

If you are a DOS PAL programmer, you can use this procedure to operate on tables by specifying the table name, rather than using a variable.

Syntax

```
1. cMin ( const tableName String, const fieldName String ) Number
2. cMin ( const tableName String, const fieldNum SmallInt ) Number
```

Example

cMin example

The following example displays the minimum value in the Total Invoice field of the *Orders* table:

```
; showMinOrder::pushButton
method pushButton(var eventInfo Event)
var
  orderTbl Table
endVar


if orderTbl.attach("Orders.db") then
    ; display smallest order in a dialog box
    msgInfo("Smallest Order in History", orderTbl.cMin("Total Invoice"))


else
  msgStop("Sorry", "Can't open Orders table.")
endIf


endMethod
```

cNpv method/procedure

Returns the net present value of a column, based on a discount or interest rate.

Syntax

```
1. cNpv ( const fieldName String, const discRate AnyType ) Number
2. cNpv ( const fieldNum SmallInt, const discRate AnyType ) Number
```

Description

cNpv returns the net present value of the column of fields specified by *fieldName* or *fieldNum*. This method respects the limits of restricted views set by **setRange** or **setGenFilter**. **cNpv** handles blank values as specified in the **blankAsZero** setting for the session.

The net present value calculation is based on *discRate*, expressed as a decimal (e.g., 0.12 for 12 percent). **cNpv** calculates net present values using the following formula:

cNpv = *sum*(*p* = 1 to *n*) of *Vp* / (1 + *r*)*p*

(*n* = number of periods, *Vp* = cash flow in *p*th period, and *r* = rate per period)

Throughout the retry period, this method attempts to place a read lock on the table. If a lock cannot be placed, the method fails.

DOS

If you are a DOS PAL programmer, you can use this procedure to operate on tables by specifying the table name, rather than using a variable.

Syntax

```
1. cNpv ( const tableName String, const fieldName String, const discRate AnyType ) Number
2. cNpv ( const tableName String, const fieldNum SmallInt, const discRate AnyType ) Number
```

Example

{button ,AL(`OPAL_TYPE_TABLE;OPAL_METH_TBCAVE;OPAL_METH_TBCMAX;OPAL_METH_TBCMIN;OPAL_METH_TBCSUM;OPAL_METH_TBCVAR;',0,"Defaultoverview",)} Related Topics

cNpv example

The following example defines a Table variable for the *GoodFund* table and calculates the net present value for the Expected Return field. The net present value is calculated based on a monthly interest rate.

```
; calcNPV::pushButton
method pushButton(var eventInfo Event)
var
  tbl Table
  goodFundNPV, apr Number
endVar
apr = .125                      ; annual percentage rate


tbl.attach("GoodFund.db")


; calculate net present value based on monthly interest rate
goodFundNPV = tbl.cNpv("Expected Return", (apr / 12))
msgInfo("Net present value", goodFundNPV)


endMethod
```

compact method

Removes deleted records from a table.

Syntax

```
compact ( [ const regIndex Logical ] ) Logical
```

Description

compact removes deleted records from a table.

Deleted records are not immediately removed from a dBASE table. Instead, they are flagged as deleted and kept in the table. The optional argument *regIndex* specifies whether to regenerate or update the indexes associated with the table. When *regIndex* is set to True, this method regenerates all indexes associated with the table. This includes indexes specified by **usesIndexes**, and the .MDX index (whose name matches the table name). If *regIndex* is set to False, indexes are not regenerated. By default, *regIndex* is set to True.

If you delete records from a Corel Paradox table, they cannot be retrieved. However, the table file and associated index files contain dead space where the record was originally stored. If you use **compact** with a Corel Paradox table, all indexes are regenerated and dead space is removed.

This method fails if any locks have been placed on the table, or the table is open. This method returns True if successful; otherwise, it returns False.

 **Example**

{button ,AL(`OPAL_TYPE_TABLE;OPAL_METH_TBSHDEL;OPAL_METH_TBUSIND;',0,"Defaultoverview",)}

Related Topics

compact example

The following example demonstrates how **compact** affects indexes specified by **usesIndexes**. In this example, the *ordTbl* Table variable is attached to ORDERS.DBF and *salesTbl* is attached to SALES.DBF. Because *ordTbl* uses INDEX1.NDX and INDEX2.NDX (specified by **usesIndexes**), **compact** regenerates INDEX1.NDX and INDEX2.NDX if *regIndex* is set to True. In this example, *regIndex* is set to False and **compact** affects only ORDERS.NDX:

```
; compactTbls::pushButton
method pushButton(var eventInfo Event)
var
  ordTbl, salesTbl Table
endVar


ordTbl.usesIndexes("index1.ndx", "index2.ndx")
ordTbl.attach("Orders.dbf")
ordTbl.compact(False)
  ; removes deleted records and fixes Orders.mdx


salesTbl.usesIndexes("index3.mdx")
salesTbl.attach("Sales.dbf")
salesTbl.compact()
  ; removes deleted records and regenerates all indexes


endMethod
```

copy method/procedure

Copies a table.

Syntax

```
1. copy ( const destTable String ) Logical
2. copy ( const destTable Table ) Logical
```

Description

copy copies the records from a source table to a target table specified in *destTable*. The data from the source table completely replaces the data in target table. The source and target tables can be different table types. If the target table is open, the method fails.

Throughout the retry period, this method attempts to place a write lock on the source table, and a full (exclusive) lock on the target table. If either lock cannot be placed, the method fails.

For more information, see Copying to a different table type in the User's Guide Help.

DOS

If you are a DOS PAL programmer, you can use this procedure to operate on tables by specifying the table name, rather than using a variable.

Syntax

```
1. copy ( const sourceTable String, const destTable String ) Logical
2. copy ( const sourceTable String, const destTable Table ) Logical
```

Example

copy example

In the following example, the **pushButton** method for *backupCust* copies the *Customer* table to *CustBak*. If *CustBak* already exists in the current directory, this code asks for confirmation before overwriting it:

```
; backupCust::pushButton
method pushButton(var eventInfo Event)
var
  srcTbl  Table
  destTbl String
endVar
destTbl = "CustBak.db"
srcTbl.attach("Customer.db")


if isTable(destTbl) then      ; if "CustBak.db" exists
                              ; ask for confirmation
  if msgQuestion("Copy table", "Overwrite " + destTbl + "?") = "Yes" then
    return
  endIf
endIf
srcTbl.copy(destTbl)    ; this copies Customer.db to CustBak.db
; Does not copy .VAL file if all it contains is RI information.
endMethod
```

create keyword

Creates a table.

Syntax

```
create tableName [ as tableType ] [ database db ]
    [ [ like likeObject ]
      [ with fieldName : type [ , fieldName : type ] * ]
      [ where fieldID is newname [ , fieldID is newname ] * ]
      [ without fieldID [ , fieldID ] * ]
      [ struct fieldStructTable ]
      [ indexStruct indexStructTable ]
      [ refIntStruct refIntStructTable ]
      [ secStruct secStructTable ]
      [ languageDriver driverName ]
      [ versionLevel versionNumber ]
      ] *
    [ key fieldID [ , fieldID ] * ]
endCreate
```

Description

create creates a table specified by *tableName*. Unless an **as** clause explicitly specifies a table type (see below), **create** uses the *tableName* extension to infer a table type (.DB is a Corel Paradox table and .DBF is a dBASE table.) For example, given Orders.dbf for *tableName*, **create** creates a dBASE table. If *tableName* does not include an extension, **create** creates a Corel Paradox table.

If *tableName* exists, **create** attempts to place a full lock on it throughout the retry period. If the lock cannot be placed, **create** fails.

The following clauses specify table attributes. They are optional, and can appear in any order within the **create** structure. The clauses are executed in the order they appear in the structure.

The **as** *tableType* clause specifies the table format:

AS "Corel Paradox"

If **as** is omitted, **create** creates a Corel Paradox table by default (unless the table resides on a SQL server. See the discussion of the **database** clause, below).

The **database** *db* clause specifies a Database variable (opened before creating the new table) that determines where the table resides. If the database is on an SQL server, the table is of a type appropriate for the server. By default, the table is created in the working directory:

DATABASE megaData

The **like** *likeObject* clause specifies an open TCursor, table name, or Table variable from which you can borrow field names, field types, the language driver, and the version level. The **like** clause does not borrow validity checks, primary or secondary indexes, referential integrity information, or security information. (Use **struct**, **indexStruct**, **refIntStruct**, and **secStruct** options to borrow more detailed information):

```
LIKE "Sales.dbf"        ; table name as a string
LIKE ordersTC           ; a TCursor variable pointing to ORDERS.DB
LIKE ordersTB           ; a Table variable pointing to ORDERS.DB
```

The **with** *fieldName* : *type* clause adds one or more fields to the table structure:

with "Last name" : "A20", "First name" : "A15", "Quantity" : "N"

You can specify the field type for *fieldName* in *type*. Valid values for *type* vary depending on the type of table you are creating. Corel Paradox tables use specific field names. Tables created on servers other than Corel Paradox require field name translations.

The following tables list valid field specifications for Corel Paradox and dBASE tables:

Corel Paradox tables	3.5 and earlier	4.5	5.0	7
Alpha	A*nnn*	A*nnn*	A*nnn*	A*nnn*
Number	N	N	N	N

Money	$	$	$	$
Date	D	D	D	D
Short	S	S	S	S
Memo	(none)	Mnnn	Mnnn	Mnnn
Formatted Memo	(none)	(none)	Fnnn	Fnnn
Binary	(none)	Bnnn	Bnnn	Bnnn
Graphic	(none)	(none)	Gnnn	Gnnn
OLE	(none)	(none)	Onnn	Onnn
Logical	(none)	(none)	L	L
Long Integer	(none)	(none)	I	
Time	(none)	(none)	T	T
Timestamp	(none)	(none)	@	@
BCD	(none)	(none)	#	#
Autoincrement	(none)	(none)	+	+
Bytes	(none)	(none)	Y	Y

dBASE tables	**III+**	**IV**	**V**
Character	Cnnn	Cnnn	Cnnn
Number	Nnnn	Nnnn	Nnnn
Date	D	D	D
Logical	L	L	L
Memo	M	M	M
Float	(none)	Fnnn.d	Fnnn.d
OLE	(none)	(none)	O
Binary	(none)	(none)	B

The **where** *fieldID* **is** "*newName*" clause changes the name of one or more fields specified by the name or number *fieldID* to *newName*:

```
where "Last name" IS "Customer last name", 2 IS "Customer first name"
```

The **without** *fieldID* clause removes one or more fields (specified by name or number) from the structure. Example:

```
without 4, "Country code"
```

The **struct** clause specifies in *fieldStructTable* an open TCursor, table name, or Table variable from which you can borrow the field-level structure. Unlike the **like** clause, **struct** borrows all validity check and primary key information. Use **enumFieldStruct** to generate *fieldStructTable* (or create it manually) before executing **create**:

```
struct "CustFlds.db"
```

The **indexStruct** clause specifies in *indexStructTable* an open TCursor, table name, or Table variable from which you can borrow secondary index information. Use **enumIndexStruct** to generate *indexStructTable* (or create it manually) before executing **create**:

```
indexStruct "CustIndx.db"
```

The **refIntStruct** clause specifies an open TCursor, table name, or Table variable from which you can borrow referential integrity information. Use **enumRefIntStruct** to generate *refIntStructTable* (or create it manually) before executing **create**:

```
refIntStruct "Cust_Ref.db"
```

The **secStruct** clause specifies in *secStructTable* an open TCursor, table name, or Table variable from which you can borrow security information. Use **enumSecStruct** to generate *secStructTable* (or create your own) before executing **create**:

```
secStruct "Cust_Sec.db"
```

When you use **secStruct**, Corel Paradox automatically protects the table with the master password *secret*. For information about master passwords, see About password security in the User's Guide help.

The **languageDriver** clause specifies in *driverName* the internal name of a language driver to use with the table. A language driver determines the table's sort order and available character set. For a list if language drivers, see the User's Guide Help topics on language drivers for Corel Paradox tables, or language drivers for dBASE tables.

The **versionLevel** clause specifies in *versionNumber* what level of table to create. Valid values for *versionNumber* are listed in the following table.

Table type	Version number
Corel Paradox	3 specifies a level 3 table corresponding to that created for Corel Paradox 3.5 and earlier (Corel Paradox Engine version 2)
	4 specifies a level 4 table corresponding to Corel Paradox for Windows 4.5 and earlier and Corel Paradox for DOS 4.0 and 4.5 (Corel Paradox Engine version 3)
	5 specifies a level 5 table corresponding to Corel Paradox for Windows 5.0
	7 specifies a level 7 table corresponding to Corel Paradox 7
dBASE	3 specifies a dBASE III table
	4 specifies a dBASE IV table
	5 specifies a dBASE for Windows table

The **key** *fieldID* clause specifies one or more key fields. You must specify key fields in order from left to right:

```
key "Last name", "First name"
```

Fields are created in the order you specify them, whether explicitly using a **with** clause, or as implied by one or more **like** clauses. **where** and **without** clauses are meaningless unless preceded by a **like** clause.

Note

- Because **create** is not a method, dot notation is inappropriate. Instead, use = to assign the **create** structure to a Table variable.

Examples

create keyword examples

Example1	Creating a new table
Example2	Using an existing table's structure

create example 1

The following example creates a Corel Paradox table named PARTS.DB. The table has three fields (Part number, Part name, and Quantity) and one key field (Part number).

```
; createParts::pushButton
method pushButton(var eventInfo Event)
var
  newParts Table
  partsTV TableView
endVar
if isTable("Parts.db") then
  if msgQuestion("Confirm",
                 "Parts.db exists. Overwrite it?") <> "Yes" then
    return
  endIf
endIf


newParts = create "Parts.db"
             WITH "Part number" : "A20",
                  "Part name" : "A20",
                  "Quantity" : "S"
             KEY "Part number"
           endCreate

partsTV.open("Parts.db")      ; Open the new table.
endMethod
```

create example 2

The following examples show two ways to create a dBASE table named NEWSALES.DBF using the same structure as the dBASE table SALES.DBF:

```
; version 1
var
  newSales Table
endVar
newSales = CREATE "Newsales.dbf"
             LIKE "Sales.dbf"
           ENDCREATE


; version 2
var
  newSales Table
  salesTC TCursor
endVar
salesTC.open("Sales.dbf")
newSales = CREATE
             LIKE salesTC
           ENDCREATE
```

The following example uses the **struct** option to borrow field-level information (including primary keys and validity checks) for use in a new table. For more information, see **enumFieldStruct**.

```
; makeNewCust::pushButton
method pushButton(var eventInfo Event)
var
  custTbl, newCustTbl Table
  custTC TCursor
endVar


custTbl.attach("Customer.db")
if custTbl.isTable() then


  if custTbl.enumFieldStruct("CustFlds.db") then


    ; Open a TCursor for CustFlds table.
    custTC.open("CustFlds.db")
    custTC.edit()


    ; This loop scans through the CustFlds table and
    ; changes ValCheck definitions for every field.
    scan custTC :
      custTC."_Required Value" = 1   ; Make all fields required.
    endScan


    ; Now create NEWCUST.DB and borrow field names,
    ; ValChecks and key fields from CUSTFLDS.DB.
    newCustTbl = CREATE "NewCust.db"
                   STRUCT "CustFlds.db"
                 ENDCREATE


    ; NEWCUST.DB requires that all fields be filled


  else
    msgStop("Error", "Can't get field structure for Customer table.")
  endIf


else
  msgStop("Error", "Can't find Customer table.")
endIf


endMethod
```

Language drivers for Corel Paradox tables

The following table displays the language drivers that you can use for Corel Paradox tables, and the code page for each driver. Use the internal name to specify *driverName*.

Note

- Internal language driver names are case-sensitive.

Driver name	Internal	Language/DOS Code Page
Corel Paradox 'ascii'	ASCII	English (US)/437
Corel Paradox 'hebrew'	HEBREW	Hebrew
Corel Paradox 'intl'	INTL	International/437
Corel Paradox 'intl850'	INTL850	International/850
Corel Paradox 'nordan'	NORDAN	Danish-Norwegian
Corel Paradox 'turk'	TURK	Turkish
Corel Paradox ANSI 'turk'	ANTURK	Turkish
Corel Paradox ANSI China	ANCHINA	Chinese
Corel Paradox ANSI Cyrillic	ANCYRR	Russian
Corel Paradox ANSI Czech	ANCZECH	Czech
Corel Paradox ANSI Greek	ANGREEK1	Greek
Corel Paradox ANSI HEBREW	ANHEBREW	ANSI Hebrew
Corel Paradox ANSI Hun DC	ANHUNDC	Hungarian
Corel Paradox ANSI Intl	ANSIINTL	ANSI International
Corel Paradox ANSI Intl850	ANSII850	ANSI International/850
Corel Paradox ANSI Korea	ANKOREA	Korean
Corel Paradox ANSI Nordan4	ANSINOR4	ANSI Danish-Norwegian/4
Corel Paradox ANSI Polish	ANPOLISH	Polish
Corel Paradox ANSI Slovene	ANSISLOV	Yugoslavia
Corel Paradox ANSI Spanish	ANSISPAN	ANSI Spanish
Corel Paradox ANSI Swedfin	ANSISWFN	ANSI Swedish-Finnish
Corel Paradox ANSI Thai	ANTHAI	ANSI Thai
Corel Paradox China 437	CHINA	Chinese/437
Corel Paradox Cyrr 866	CYRR	Russian/866
Corel Paradox Czech 852	CZECH	Czech/852
Corel Paradox Czech 867	CSKAMEN	Czech/867
Corel Paradox ESP 437	SPANISH	Spanish/437
Corel Paradox Greek GR437	GRCP437	Greek/437
Corel Paradox Hun 852 DC	HUN852DC	Hungarian/852
Corel Paradox ISL 861	ICELAND	Iceland/861
Corel Paradox Korea 949	KOREA	Korean/949
Corel Paradox NORDAN	NORDAN	Danish-Norwegian/865
Corel Paradox NORDAN40	NORDAN40	Danish-Norwegian/865
Corel Paradox Polish 852	POLISH	Polish/852
Corel Paradox Slovene 852	SLOVENE	Yugoslavia/852
Corel Paradox SWEDFIN	SWEDFIN	Swedish-Finnish/437
Corel Paradox Thai 437	THAI	Thai/437

Language drivers for dBASE tables

The following table displays the language drivers that you can use for dBASE tables. Use the internal name to specify *driverName*.

Note

- Internal language driver names are case-sensitive.

Driver	Internal name	Language
dBASE CHN pc437	DB437CN0	Chinese
dBASE CSY cp852	DB852CZ0	Czech
dBASE CSY cp867	DB867CZ0	Czech
dBASE DAN cp865	DB865DA0	Danish
dBASE DEU cp437	DB437DE0	German
dBASE DEU cp850	DB850DE0	German
dBASE ELL GR437	DB437GR0	Greek
dBASE ENG cp437	DB437UK0	English (U.K)
dBASE ENG cp850	DB850UK0	English (U.K)
dBASE ENU cp437	DB437US0	English (U.S.)
dBASE ENU cp850	DB850US0	English (U.S.)
dBASE ESP cp437	DB437ES1	Spanish
dBASE ESP cp850	DB850ES0	Spanish
dBASE FIN cp437	DB437FI0	Finnish
dBASE FRA cp437	DB437FR0	French
dBASE FRA cp850	DB850FR0	French
dBASE FRC cp850	DB850CF0	French (Can.)
dBASE FRC cp863	DB863CF1	French (Can.)
dBASE HUN cp852	DB852HDC	Hungarian
dBASE ITA cp437	DB437IT0	Italian
dBASE ITA cp850	DB850IT0	Italian
dBASE KOR cp949	DB949KO0	Korean
dBASE NLD cp437	DB437NL0	Dutch
dBASE NLD cp850	DB850NL0	Dutch
dBASE NOR cp437	DB437NO0	Norwegian
dBASE NOR cp865	DB865NO0	Norwegian
dBASE PLK pc852	DB852PO0	Polish
dBASE PTB cp850	DB850PT0	Portuguese (Bra.)
dBASE PTG cp860	DB860PT0	Portuguese
dBASE RUS cp866	DB866RU0	Russian
dBASE SLO cp852	DB852SL0	Yugoslavian
dBASE SVE cp437	DB437SV0	Swedish
dBASE SVE cp850	DB850SV0	Swedish
dBASE TRK cp857	DB857TR0	Turkish
dBASE TWN cp437	DB437TW0	Taiwanese

Field translations for tables

The following table displays the field names used in tables that are created on dBASE, Oracle, Sybase, InterBase and Informix servers:

Corel Paradox	dBASE	Interbase	Oracle	Sybase	Informix
Alpha	Character	Varying	Character	VarChar	Character
Number	Float{20.4}	Double	Number	Float	Float
Money	Float{20.4}	Double	Number	Money	Money{16.2}
Date	Date	Date	Date	DateTime	Date
Short	Number{6.0}	Short	Number	SmallInt	SmallInt
Memo	Memo	Blob/1	Long	Text	Text
Binary	Memo	Blob	LongRaw	Image	Byte
Formatted Memo	Memo	Blob	LongRaw	Image	Byte
OLE	Memo	Blob	LongRaw	Image	Byte
Graphic	Memo	Blob	LongRaw	Image	Byte
Long	Number{11.0}	Long	Number	Int	Integer
Time	Character{>8}	Character{>8}	Character{>8}	Character{>8}	Character{>8}
DateTime	Character{>8}	Date	Date	DateTime	DateTime
Bool	Bool	Character{1}	Character{1}	Bit	Character
AutoInc	Number{11.0}	Long	Number	Int	Integer
Bytes	Bytes	Blob	LongRaw	Image	Byte
BCD	N/A	N/A	N/A	N/A	N/A

createIndex method

Creates an index for a table.

Syntax

```
1. createIndex ( const attrib DynArray[ ] AnyType, const fieldNames Array[ ] String ) Logical
2. createIndex ( const attrib DynArray[ ] AnyType, const fieldNums Array[ ] SmallInt ) Logical
```

Description

createIndex creates an index using attributes specified in a DynArray named *attrib* and the field names (or numbers) specified in an Array named *fieldNames* (or *fieldNums*). This method is provided as an alternative to the **index** structure. It is especially useful when you don't know the index structure beforehand (e.g., when the information is supplied by the user).

Each key of the DynArray must be a string. You do not have to include all the keys to use **createIndex**. Any key you omit is assigned the corresponding default value.

The following table displays the key strings and their corresponding values:

String value	Description
MAINTAINED	If True, the index is incrementally maintained. That is, after a table is changed, only that portion of the index affected by the change is updated. If False, Corel Paradox does not maintain the index automatically. Maintained indexes typically result in better performance. Default = False (Corel Paradox tables only).
PRIMARY	If True, the index is a primary index. If False, it's a secondary index. Default = False (Corel Paradox tables only).
CASEINSENSITIVE	If True, the index ignores differences in case. If False, it considers case. Default = False (Corel Paradox tables only).
DESCENDING	If True, the index is sorted in descending order, from highest values to lowest. If False, it is sorted in ascending order. Default = False.
UNIQUE	If True, records with duplicate values in key fields are ignored. If False, duplicates are included and available.
IndexName	A name used to identify this index. No default value, unless you're creating a secondary, case-sensitive index on a single field, in which case the default value is the field name. For dBASE tables, the index name must be a valid DOS filename. If you do not specify an extension, .NDX is added automatically.
TagName	The name of the index tag associated with the index specified in *indexName* (dBASE tables only).

For more information on indexes, see About keys and indexes in tables in the User's Guide Help.

Examples

{button ,AL(`OPAL_TYPE_TABLE;OPAL_METH_TBDROP;OPAL_METH_TBINDE;OPAL_METH_TBSIND;OPAL_METH_TBUSIND;',0,"Defaultoverview",)} Related Topics

createIndex method examples

createIndex example 1

The following example builds a maintained secondary index for a Corel Paradox table named CUSTOMER.DB. If the *Customer* table cannot be found or locked, **createIndex** aborts the operation.

```
method pushButton(var eventInfo Event)
var
   stTbName      String
   tbCust         Table
   arFieldNames   Array[3] String
   dyAttrib      DynArray[]AnyType
endVar


stTbName = "Customer.db"


arFieldNames[1] = "Customer No"
arFieldNames[2] = "Name"
arFieldNames[3] = "Street"


dyAttrib["PRIMARY"]    = False
dyAttrib["MAINTAINED"] = True
dyAttrib["IndexName"]  = "NumberNameStreet"


if isTable(stTbName) then
  tbCust.attach(stTbName)
   if not tbCust.lock("FULL") then
      errorShow()
      return
   endIf


   if not tbCust.createIndex(dyAttrib, arFieldNames) then
      errorShow()
   endIf


; This createIndex statement has the same effect
; as the following INDEX structure:
{
  INDEX tbCust                  ; Create index for Customer.db.
     MAINTAINED
    ON "Customer No", "Name", "Street"
  ENDINDEX
}


else
  errorShow()
endIf


endMethod
```

createIndex example 2

The following example adds a unique index tag named StatProv to the production index for a dBASE table named CUSTOMER.DBF:

```
method pushButton(var eventInfo Event)
var
   tbCust         Table
   arFieldNames   Array[1] String
   dyAttrib      DynArray[]AnyType
endVar


arFieldNames[1] = "STATE_PROV"


dyAttrib["UNIQUE"]     = True
dyAttrib["MAINTAINED"] = True


; A dBASE index name must be a valid DOS filename.
; If an extension is omitted, .NDX is appended automatically.


dyAttrib["IndexName"] = "Customer.Mdx"
dyAttrib["TagName"] = "StatProv"


if isTable("Customer.dbf") then
  tbCust.attach("Customer.dbf")


  if not tbCust.createIndex(dyAttrib, arFieldNames) then
       errorShow()
  endIf


; This createIndex statement has the same effect
; as the following INDEX structure:
{
  INDEX tbCust                        ; Create index for Customer.dbf.
       UNIQUE
    ON "STATE_PROV"                   ; Index on this field.
    TAG "StatProv" OF "Customer.dbf"  ; Name the tag "StatProv".
  ENDINDEX
}


else
  errorShow()
endIf


endMethod
```

cSamStd method/procedure

Returns the sample standard deviation of a table's column.

Syntax

```
1. cSamStd ( const fieldName String ) Number
2. cSamStd ( const fieldNum SmallInt ) Number
```

Description

cSamStd returns the sample standard deviation for the column of numeric fields specified by *fieldName* or *fieldNum*. This method respects the limits of restricted views displayed in a linked table frame or multi-record object. **cSamStd** handles blank values as specified in the **blankAsZero** setting for the session.

The sample standard deviation calculation is based on the sample variance and uses the following formula:

sqrt((*sampVar*) * (*n*/(*n*-1)))

(*sampVar* = cSamVar(*tableName*, *fieldName*) and *n* = cCount(*tableName*, *fieldName*)

Throughout the retry period, this method attempts to place a read lock on the table. If a lock cannot be placed, the method fails.

The population standard deviation is calculated using the **cStd** method.

DOS

If you are a DOS PAL programmer, you can use this procedure to operate on tables by specifying the table name, rather than using a variable.

Syntax

```
1. cSamStd ( const tableName String, const fieldName String ) Number
2. cSamStd ( const tableName String, const fieldNum SmallInt ) Number
```

Example

{button ,AL(`OPAL_TYPE_TABLE;OPAL_METH_TBCSVA;OPAL_METH_TBCAVE;OPAL_METH_TBCCNT;OPAL_METH_TBCMAX;OPAL_METH_TBCMIN;OPAL_METH_TBCNPV;OPAL_METH_TBCSUM;OPAL_METH_TBCVAR;OPAL_METH_TBCSTD;',0,"Defaultoverview",)} Related Topics

cSamStd example

The following example calculates the sample standard deviation of test scores for the Winter quarter. This code is attached to the **pushButton** method for *showSamStd*:

```
; showSamStd::pushButton
method pushButton(var eventInfo Event)
   const
      kTbName = "winter"
   endConst

   var
      tbWinter   Table
      nuSamStd    Number
   endVar


   tbWinter.attach(kTbName)
   nuSamStd = tbWinter.cSamStd("TestScore")
   nuSamStd.view()
endMethod
```

cSamVar method/procedure

Returns the sample variance of a table's column.

Syntax

```
1. cSamVar ( const fieldName String ) Number
2. cSamVar ( const fieldNum SmallInt ) Number
```

Description

cSamVar returns the sample variance for the column of fields specified by *fieldName* or *fieldNum*. This method respects the limits of restricted views set by **setRange** or **setGenFilter**. **cSamVar** handles blank values as specified in the **blankAsZero** setting for the session.

The sample variance is calculated using the formula:

cVar(*tableName*, *fieldName*) * (*n*/(*n - 1*))

(n = cCount(*tableName, fieldName*))

Throughout the retry period, this method attempts to place a read lock on the table. If a lock cannot be placed, the method fails.

DOS

If you are a DOS PAL programmer, you can use this procedure to operate on tables by specifying the table name, rather than using a variable.

Syntax

```
1. cSamVar ( const tableName String, const fieldName String ) Number
2. cSamVar ( const tableName String, const fieldNum SmallInt ) Number
```

Example

{button ,AL(`OPAL_TYPE_TABLE;OPAL_METH_TBCAVE;OPAL_METH_TBCCNT;OPAL_METH_TBCMAX;OPAL_METH_TBCMIN;OPAL_METH_TBCNPV;OPAL_METH_TBCSST;OPAL_METH_TBCSTD;OPAL_METH_TBCSUM;OPAL_METH_TBCVAR;',0,"Defaultoverview",)} **Related Topics**

cSamVar example

The following example calculates the sample variance of two fields in the *Answer* table. This code is attached to the **pushButton** method for *showSamVar*.

```
; showSamVar::pushButton
method pushButton(var eventInfo Event)
var
   empTbl Table
   tblName String
   calcSalary, calcYears Number
endVar
tblName = "Answer"


empTbl.attach(tblName)
calcSalary = empTbl.cSamVar("Salary")  ; get sample variance for Salaries
calcYears  = empTbl.cSamVar(2)         ; assume "Years in service" is field 2
msgInfo("Sample Variance",             ; display info in a dialog box
        "Salaries : " + String(calcSalary,
        "\nYears in service : ", calcYears))


endMethod
```

cStd method/procedure

Returns the standard deviation of the values in a column.

Syntax

```
1. cStd ( const fieldName String ) Number
2. cStd ( const fieldNum SmallInt ) Number
```

Description

cStd returns the population standard deviation of the column of fields specified by *fieldName* or *fieldNum*. This method respects the limits of restricted views set by **setRange** or **setGenFilter**. This method handles blank values as specified in the **blankAsZero** setting for the session. Population standard deviation calculations are based on the variance. For more information, see **cVar**.

Throughout the retry period, this method attempts to place a read lock on the table. If a lock cannot be placed, the method fails.

DOS

If you are a DOS PAL programmer, you can use this procedure to operate on tables by specifying the table name, rather than using a variable.

Syntax

```
1. cStd ( const tableName String, const fieldName String ) Number
2. cStd ( const tableName String, const fieldNum SmallInt ) Number
```

Example

{button ,AL(`OPAL_TYPE_TABLE;OPAL_METH_TBCSST;OPAL_METH_TBCSVA;OPAL_METH_TBCSUM;OPAL_METH_TBCAVE;OPAL_METH_TBCCNT;OPAL_METH_TBCMAX;OPAL_METH_TBCMIN;OPAL_METH_TBCNPV;OPAL_METH_TBCVAR;',0,"Defaultoverview",)} **Related Topics**

cStd example

In the following example, the **pushButton** method for *thisButton* calculates the population standard deviation for two separate fields and displays the results in a dialog box:

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
  myTable Table
  test1, test2 Number
endVar
myTable.attach("scores.db")
test1 = myTable.cStd("Test1")
test2 = myTable.cStd(2)         ; assumes Test2 is field 2
msgInfo("Standard Deviation",
        "Test1 results : " + String(test1) + "\n" +
        "Test2 results : " + String(test2))
endMethod
```

cSum method/procedure

Returns the sum of the values in of a table's column.

Syntax

```
1. cSum ( const fieldName String ) Number
2. cSum ( const fieldNum SmallInt ) Number
```

Description

cSum returns the sum of the values in the column of fields specified by *fieldName* or *fieldNum*. This method respects the limits of restricted views set by **setRange** or **setGenFilter**. **cSum** handles blank values as specified in the **blankAsZero** setting for the session.

Throughout the retry period, this method attempts to place a read lock on the table. If a lock cannot be placed, the method fails.

DOS

If you are a DOS PAL programmer, you can use this procedure to operate on tables by specifying the table name, rather than using a variable.

Syntax

```
1. cSum ( const tableName String, const fieldName String ) Number
2. cSum ( const tableName String, const fieldNum SmallInt ) Number
```

Example

{button ,AL(`OPAL_TYPE_TABLE;OPAL_METH_TBCAVE;OPAL_METH_TBCCNT;OPAL_METH_TBCMAX;OPAL_METH_TBCMIN;OPAL_METH_TBCNPV;OPAL_METH_TBCSST;OPAL_METH_TBCSVA;OPAL_METH_TBCSTD;OPAL_METH_TBCVAR;',0,"Defaultoverview",)} Related Topics

cSum example

In the following example, the **pushButton** method for *sumOrders* uses both forms of **cSum** syntax to calculate totals for two fields in ORDERS.DB:

```
; sumOrders::pushButton
method pushButton(var eventInfo Event)
var
  orderTbl Table
  orderTotal, amtPaid Number
  tblName String
endVar
tblName = "Orders"


orderTbl.attach(tblName)
orderTotal = orderTbl.cSum("Total Invoice")
amtPaid    = orderTbl.cSum(7)    ; assumes Amount Paid is field 7
msgInfo("Order Totals",
        "Total Orders : " + String(orderTotal) + "\n" +
        "Total Receipts : " + String(amtPaid))


endMethod
```

cVar method/procedure

Returns the variance of a field in a table.

Syntax

```
1. cVar ( const fieldName String ) Number
2. cVar ( const fieldNum SmallInt ) Number
```

Description

cVar returns the population variance of the column of fields specified by *fieldName* or *fieldNum*. This method respects the limits of restricted views set by **setRange** or **setGenFilter**. **cVar** handles blank values as specified in the **blankAsZero** setting for the session.

Throughout the retry period, this method attempts to place a read lock on the table. If a lock cannot be placed, the method fails.

DOS

If you are a DOS PAL programmer, you can use this procedure to operate on tables by specifying the table name, rather than using a variable.

Syntax

```
1. cVar ( const tableName String, const fieldName String ) Number
2. cVar ( const tableName String, const fieldNum SmallInt ) Number
```

Example

{button ,AL(`OPAL_TYPE_TABLE;OPAL_METH_TBCAVE;OPAL_METH_TBCCNT;OPAL_METH_TBCMAX;OPAL_METH_TBCMIN;OPAL_METH_TBCNPV;OPAL_METH_TBCSVA;OPAL_METH_TBCSST;OPAL_METH_TBCSUM;',0,"Defaultoverview",)} Related Topics

cVar example

In the following example, the **pushButton** method for *thisButton* calculates the population variance deviation for two separate fields and displays the results in a dialog box:

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
  myTable Table
  test1, test2 Number
endVar
myTable.attach("scores.db")
test1 = myTable.cVar("Test1")
test2 = myTable.cVar(2)       ; assumes Test2 is field 2
msgInfo("Population Variance",
        "Test1 results : " + String(test1) + "\n" +
        "Test2 results : " + String(test2))

endMethod
```

delete method/procedure

Deletes a table.

Syntax

```
delete ( ) Logical
```

Description

delete deletes a table without asking for confirmation. Compare this method to **empty**, which removes data from a table but does not delete it.

If the table is open or is locked, **delete** fails.

DOS

If you are a DOS PAL programmer, you can use this procedure to operate on tables by specifying the table name, rather than using a variable.

Syntax

```
delete ( const tableName String ) Logical
```

Example
{button ,AL(`OPAL_TYPE_TABLE;OPAL_METH_TBEMPT;',0,"Defaultoverview",)} Related Topics

delete example

The following example deletes ANSWER.DB from the private directory:

```
; delAnswer::pushButton
method pushButton(var eventInfo Event)
var
  tbl Table
  tblName String
endVar


tblName = privDir() + "\\Answer.db"


tbl.attach(tblName)
if tbl.isTable() then
  tbl.delete()
  message(tblName, " deleted.")
else
  message("Can't find ", tblName, ".")
endIf


endMethod
```

dropGenFilter method

Removes the filter criteria associated with a Table variable.

Syntax

```
dropGenFilter ( ) Logical
```

Description

dropGenFilter removes the filter criteria associated with a Table variable. Any indexes and ranges remain in effect in the unfiltered table.

Example

{button ,AL(`OPAL_TYPE_TABLE;OPAL_METH_TBGETGENFILTER;OPAL_METH_TBSETGENFILTER;OPAL_METH_TCDROPGENFILTER;OPAL_METH_UIDROPGENFILTER;OPAL_INFO_TBUSINGRANGESANDFILTERS;',0,"Defaultoverview",)} Related Topics

dropGenFilter example

In the following example, a form contains a button named *btnCACustomers*. The **pushButton** method for *btnCACustomers* attaches a Table variable to the *Customer* table, sets filter criteria, and stores the value in the number variable *nSubTotal*. **dropGenFilter** removes the filter and the total number of records is stored in a number variable named *nTotal*. Finally, a message information box displays the number of customers in California compared to the total number of customers.

```
;btnCACustomers :: pushButton
method pushButton(var eventInfo Event)
   var
      tbl            Table
      dyn            DynArray[] AnyType
      nTotal,
      nSubTotal   Number
   endVar


   tbl.attach("CUSTOMER.DB")


   dyn["State/Prov"] = "CA"
   tbl.setGenFilter(dyn)
   nSubTotal = tbl.cCount("State/Prov")   ;Get customers in CA.


   tbl.dropGenFilter()
   nTotal = tbl.nRecords()         ;Get all customers.


   msgInfo("Customer Analysis",  string(nSubtotal) + " out of " + string(nTotal) + " reside in
California.")
endMethod
```

dropIndex method

Deletes a specified index file or tag.

Syntax

```
1. (Corel Paradox tables) dropIndex ( const indexName String ) Logical

2. (dBASE tables) dropIndex ( const indexName String
                        [ , const tagName String ] ) Logical
```

Description

dropIndex deletes a specified index file or tag.

In a Corel Paradox table, *indexName* specifies a secondary index. If you specify an empty string in *indexName*, the primary index is removed.

In a dBASE table, *indexName* specifies an .NDX file. You can also use *indexName* and *tagName* to specify an .MDX file and an index tag.

You must call **setExclusive** before calling **dropIndex** to obtain exclusive rights to the table.

dropIndex fails if the index you're trying to delete is in use, or if the table is open.

For more information about indexes, see About keys and indexes in tables in the User's Guide Help.

Example

{button ,AL(`OPAL_TYPE_TABLE;OPAL_METH_TBSIND;OPAL_METH_TBUSIND;',0,"Defaultoverview",)}

Related Topics

dropIndex example

In the following example, the **pushButton** method for *thisButton* deletes the CustName tag from an .MDX file:

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
  salesTbl Table
endVar


salesTbl.attach("Sales.dbf")          ; Sales.dbf is a dBASE table
if isTable(salesTbl) then             ; if salesTbl is a table


  ; Get exclusive access to the table.
  salesTbl.setExclusive(Yes)
  ; delete CustName tag from index2.mdx file
  if salesTbl.dropIndex("index2.mdx", "CustName") then
    msgInfo("Status", "CustName index deleted.")
  else
    msgInfo("Error", "Can't drop CustName from Index2.")
  endIf


else
  msgStop("Stop!", "Could not find Sales.dbf table.")
endIf


endMethod
```

empty method/procedure

Removes all records from a table.

Syntax

```
empty ( ) Logical
```

Description

empty removes all records from a table without asking for confirmation. This operation cannot be undone. This method returns True if it succeeds; otherwise, it returns False.

empty removes information from the table, but does not delete the table itself. Compare this method to **delete**, which does delete the table.

empty first tries to gain exclusive rights to the table. If it can't, it tries to place a write lock on the table.

If **empty** gains exclusive rights, it deletes all records in the table at once. If a write lock is placed on the table, **empty** must delete each record individually.

If **empty** gains exclusive rights to a dBASE table, all records are deleted and the table is compacted. If a write lock is placed on the table, this method flags all records as deleted, but does not remove them from the table. (Records can be undeleted from a dBASE table if they have not been removed with the **compact** method.)

DOS

If you are a DOS PAL programmer, you can use this procedure to operate on tables by specifying the table name, rather than using a variable.

Syntax

```
empty ( const tableName String ) Logical
```

Example

{button ,AL(`OPAL_TYPE_TABLE;OPAL_METH_TBDELETE;',0,"Defaultoverview",)} Related Topics

empty example

The following example prompts the user for confirmation before deleting all records from the *Scratch* table:

```
; tblEmpty::pushButton
method pushButton(var eventInfo Event)
var
  tblName String
  tblVar Table
endVar
tblName = "Scratch.db"


tblVar.attach(tblName)
if isTable(tblName) then
  if msgQuestion("Empty?", "Empty " + tblName + " ?") = "Yes" then


    if tblVar.empty() then
      message("All " + tblName + " records have been deleted.")
    else
      errorShow()
    endIf


  endIf
else
  errorShow()
endIf
endMethod
```

enumFieldNames method

Fills an array with the table's field names.

Syntax

```
enumFieldNames ( var fieldArray Array[ ] String ) Logical
```

Description

enumFieldNames fills an array named *fieldArray* with a table's field names. You must declare *fieldArray* as a resizeable array before calling this method. If *fieldArray* already exists, **enumFieldNames** overwrites it without asking for confirmation.

Example

{button ,AL(`OPAL_TYPE_TABLE;OPAL_METH_TBENUMFIELDNAMESININDEX;OPAL_METH_TBENUMFIELDSTRUCT;OPAL_METH_TBENUMINDEXSTRUCT;OPAL_METH_TBENUMREFINTSTRUCT;OPAL_METH_TBENUMSECSTRUCT;',0,"Defaultoverview",)} Related Topics

enumFieldNames example

In the following example, the **pushButton** method for the *btnEnumFields* button stores field names in a resizeable array and uses **view** to display the contents of the array:

```
; btnEnumFields::pushButton
method pushButton(var eventInfo Event)
var
  tbl Table
  arFieldNames Array[] AnyType
endVar


tbl.attach("Sales.dbf")
if tbl.isTable() then
  tbl.enumFieldNames(arFieldNames)
  arFieldNames.view()
else
  errorShow()
endIf


endMethod
```

enumFieldNamesInIndex method

Fills an array with a table index's field names.

Syntax

1. (Corel Paradox tables) **enumFieldNamesInIndex (** [const ***indexName*** String,] var ***fieldArray*** Array[ ] String **)** Logical

2. (dBASE tables) **enumFieldNamesInIndex (** [const ***indexName*** String, [const ***tagName*** String,]] var ***fieldArray*** Array[ ] String **)** Logical

Description

enumFieldNamesInIndex fills an array named *fieldArray* with the names of the fields in a table's index, as specified in *indexName*. You must declare *fieldArray* as a resizeable array before calling this method. If *fieldArray* already exists, this method overwrites it without asking for confirmation.

In a dBASE table, the argument *tagName* is required to specify an index tag within an .MDX file.

By default, *indexName* corresponds to the index currently being used.

For more information on indexes, see About keys and indexes in tables in the User's Guide Help.

Example

{button ,AL(`OPAL_TYPE_TABLE;OPAL_METH_TBENUMFIELDNAMES;OPAL_METH_TBENUMFIELDSTRUCT;OPAL_METH_TBENUMINDEXSTRUCT;OPAL_METH_TBENUMREFINTSTRUCT;OPAL_METH_TBENUMSECSTRUCT;',0,"Defaultoverview",)} Related Topics

enumFieldNamesInIndex example

In the following example, the **pushButton** method for the *showIndexFlds* button stores field names in a resizeable array and uses **view** to display the array's contents:

```
; showIndexFlds::pushButton
method pushButton(var eventInfo Event)
var
  tbl Table
  fieldNames Array[] String
endVar


tbl.attach("Sales.dbf")
if tbl.isTable() then
  tbl.enumFieldNamesInIndex("DateIndx", "byDate", fieldNames)
  ; display the index field names for byDate in DateIndx
  fieldNames.view()
else
  msgStop("Stop", "Couldn't find Sales.dbf.")
endIf


endMethod
```

enumFieldStruct method

Lists a table's field structure.

Syntax

```
1. enumFieldStruct ( const tableName String ) Logical
2. enumFieldStruct ( inMem TCursor ) Logical
```

Description

enumFieldStruct lists the field structure of a Table variable. Syntax 1 creates a Corel Paradox table; Syntax 2 stores the information in a TCursor variable.

Syntax 1 creates a Corel Paradox table *tableName*. If *tableName* already exists, this method overwrites it without asking for confirmation. If *tableName* is open, this method fails. You can also include an alias or path in *tableName*. If an alias or path is not specified, Corel Paradox creates the table in the working directory. You can supply *tableName* to the **struct** option in a **create** statement to borrow that table's field structure (including primary keys and validity checks) for use in the new table.

In Syntax 2, the structure information is stored in the TCursor variable *inMem* that you pass as an argument. Syntax 2 results in faster performance because the information is stored in system memory.

The following table displays the structure of the table in Syntax 1 or the TCursor in Syntax 2:

Field	Type	Description
Field Name	A31	Specifies the name of field
Type	A31	Specifies the data type of field
Size	S	Specifies the size of field
Dec	S	Specifies the number of decimal places, or 0 if field type doesn't support decimal places
Key	A1	Specifies whether the field is a key (* = key field, blank = not key field)
_Required Value	A1	Specifies whether the field is required (T = required, N (or blank) = Not required)
_Min Value	A255	Specifies the field's minimum value
_Max Value	A255	Specifies the field's maximum value
_Default Value	A255	Specifies the field's default value
_Picture Value	A175	Specifies the field's picture
_Table Lookup	A255	Specifies the name of lookup table (including the full path if the lookup table is not in :WORK:)
_Table Lookup Type	A1	Specifies the type of lookup table 0 (or blank) = no lookup table, 1 = Current field + private 2 = All corresponding + no help 3 = Just current field + help and field 4 = All corresponding + help
_Invariant Field ID	S	Specifies the field's ordinal position in table (first field = 1, second field = 2, etc.)

Once *tableName* is created, you can modify values in the table and use it with the **struct** option in the **create** command.

Example

enumFieldStruct example

The following example assumes that you want a new table named *NewCust* that is similar to the *Customer* table. It also assumes that you want all of the fields in *NewCust* to be required fields. The following code uses **enumFieldStruct** to load a new table (CUSTFLDS.DB) with the field-level information from *Customer*. The code then scans *CustFlds* and modifies the field definitions so that each record describes a required field. *CustFlds* is then supplied in the **struct** clause of a **create** statement.

```
; makeNewCust::pushButton
method pushButton(var eventInfo Event)
var
  custTbl, newCustTbl Table
  custTC TCursor
endVar


custTbl.attach("Customer.db")
if custTbl.isTable() then


  if custTbl.enumFieldStruct("CustFlds.db") then


    ; Open a TCursor for CustFlds table.
    custTC.open("CustFlds.db")
    custTC.edit()


    ; This loop scans through the CustFlds table and
    ; changes ValCheck definitions for every field .
    scan custTC :
      custTC."_Required Value" = 1    ; Make all fields required.
    endScan


    ; Now create NEWCUST.DB and borrow field names,
    ; ValChecks and key fields from CUSTFLDS.DB.
    newCustTbl = CREATE "NewCust.db"
                   STRUCT "CustFlds.db"
                 endCreate


    ; NEWCUST.DB requires that all fields be filled.


  else
    msgStop("Error", "Can't get field structure for Customer table.")
  endIf


else
  msgStop("Error", "Can't find Customer table.")
endIf


endMethod
```

enumIndexStruct method

Lists the structure of a table's secondary indexes.

Syntax

```
1. enumIndexStruct ( const tableName String ) Logical
2. enumIndexStruct ( inMem TCursor ) Logical
```

Description

enumIndexStruct lists the structure of a table's secondary indexes. Syntax 1 creates a Corel Paradox table; Syntax 2 stores the information in a TCursor variable.

Syntax 1 creates the Corel Paradox table specified in *tableName*. For dBASE tables, this method lists the structure of the indexes associated with the table by the **usesIndexes** method. If *tableName* already exists, this method overwrites it without asking for confirmation. You can also include an alias or path in *tableName*. If an alias or path is not specified, Corel Paradox creates the table in the working directory. You can supply *tableName* to the **indexStruct** option in a **create** statement to borrow that table's field structure (including primary keys and validity checks) for use in the new table.

In Syntax 2, the structure information is stored in the TCursor variable *inMem* that you pass as an argument. Syntax 2 results in faster performance because the information is stored in system memory.

The following table displays the structure of the table in Syntax 1 or the TCursor in Syntax 2:

Field	Type	Description
infoHeader	A1	Specifies whether this record is a header for (and the data it contains is shared by) subsequent consecutive records that have a value of N in this field
szName	A255	Specifies the index name, including path
szTagName	A31	Specifies the tag name, no path (dBASE only)
szFormat	A31	Specifies the optional index type, e.g., BTREE, HASH
bPrimary	A1	Specifies whether the index is primary
bUnique	A1	Specifies whether the index is unique
bDescending	A1	Specifies whether the index is descending
bMaintained	A1	Specifies whether the index is maintained
bCaseInsensitive	A1	Specifies whether the index is case-sensitive
bSubset	A1	Specifies whether the index is a subset index (dBASE only)
bExpIdx	A1	Specifies whether the index is an expression index (dBASE only)
iKeyExpType	N	Specifies the key type of index expression (dBASE only)
szKeyExp	A220	Specifies the key expression for expression index (dBASE only)
szKeyCond	A220	Specifies the subset condition for subset index (dBASE only)
FieldNo	N	Specifies the ordinal position of key field in table
FieldName	A31	Specifies the name of key field
bDescendingField	A1	Specifies whether the field is indexed in descending order
iIndexId	N	Specifies the ID of the index (generated by BDE) Used by restructure to specify the addition, modification or deletion of an index

tableName also includes information for indexes that are used if the dBASE table is open. To specify which indexes to associate to a Table variable, use the **usesIndexes** method and call **enumIndexStruct** to create a table that list those indexes.

For more information on indexes, see About keys and indexes in the User's Guide Help.

Example

{button ,AL(`OPAL_TYPE_TABLE;OPAL_METH_TBENUMFIELDNAMES;OPAL_METH_TBENUMFIELDNAMESININDEX;OPAL_METH_TBENUMFIELDSTRUCT;OPAL_METH_TBENUMREFINTSTRUCT;OPAL_METH_TBENUMSECSTRUCT;',0,"Defaultoverview",)} Related Topics

enumIndexStruct example

The following example assumes that you want a new table named *NewCust* that is similar to the *Customer* table. It also assumes that you don't want to borrow referential integrity or security information. The following code uses **enumFieldStruct** and **enumIndexStruct** to generate two tables (CUSTFLDS.DB and CUSTINDX.DB). *CustFlds* and *CustIndx* are then supplied to the **struct** and **indexStruct** clauses of a **create** statement.

```
; makeNewCust::pushButton
method pushButton(var eventInfo Event)
var
  custTbl, newCustTbl Table
  custTC TCursor
endVar


custTbl.attach("Customer.db")
if custTbl.isTable() then


  custTbl.enumFieldStruct("CustFlds.db")
  custTbl.enumIndexStruct("CustIndx.db")


  ; Now create NEWCUST.DB.
  ; Borrow field names, ValChecks, and key fields from CUSTFLDS.DB.
  ; Borrow secondary indexes from CUSTINDX.DB.
  newCustTbl = CREATE "NewCust.db"
                 STRUCT "CustFlds.db"
                 INDEXSTRUCT "CustIndx.db"
               ENDCREATE


else
  msgStop("Error", "Can't find Customer table.")
endIf


endMethod
```

enumRefIntStruct method

Lists a table's referential integrity information.

Syntax

```
1. enumRefIntStruct ( const tableName String ) Logical
2. enumRefIntStruct ( inMem TCursor ) Logical
```

Description

enumRefIntStruct lists referential integrity information for a Table variable. Syntax 1 creates a Corel Paradox table; Syntax 2 stores the information in a TCursor variable.

Syntax 1 creates the Corel Paradox table specified in *tableName*. If *tableName* is open, this method fails. If *tableName* already exists, this method overwrites it without asking for confirmation. You can also include an alias or path in *tableName*. If an alias or path is not specified, Corel Paradox creates the table in the working directory. You can supply *tableName* to the **refIntStruct** option in a **create** statement to borrow that table's field structure (including primary keys and validity checks) for use in the new table.

In Syntax 2, the structure information is stored in the TCursor variable *inMem* that you pass as an argument. Syntax 2 results in faster performance because the information is stored in system memory.

The following table displays the structure of the table in Syntax 1 or the TCursor in Syntax 2:

Field name	Type	Description
infoHeader	A1	Specifies whether the record is a header for (and the data it contains is shared by) subsequent consecutive records that have a value of N in this field
RefName	A31	Specifies the name to identify this referential integrity constraint
OtherTable	A255	Specifies the name (including path) of the other table in the referential integrity relationship
Slave	A1	Specifies whether the table is slave, not master (i.e., the table is dependent)
Modify	A1	Specifies the update rule (Y = Cascade, blank = Prohibit)
Delete	A1	Specifies the delete rule (blank = Prohibit). Corel Paradox does not support cascading deletes for Corel Paradox or dBASE tables.
FieldNo	N	Specifies the ordinal position of the field in this table involved in a referential integrity relationship
aiThisTabField	A31	Specifies the name of the field in this table involved in a referential integrity relationship
Other FieldNo	N	Specifies the ordinal position of the field in the other table involved in a referential integrity relationship
aiOthTabField	A31	Specifies the name of the field in the other table involved in a referential integrity relationship

Example

{button ,AL(`OPAL_TYPE_TABLE;OPAL_METH_TBENUMFIELDNAMES;OPAL_METH_TBENUMFIELDNAMESINDEX;OPAL_METH_TBENUMFIELDSTRUCT;OPAL_METH_TBENUMINDEXSTRUCT;OPAL_METH_TBENUMSECSTRUCT;',0,"Defaultoverview",)} **Related Topics**

enumRefIntStruct example

The following example uses **enumRefIntStruct** to write CUSTOMER.DB referential integrity information to the *CustRef* table. The code supplies *CustRef* to the **refIntStruct** clause in a **create** statement. When using the referential integrity structure from another table, you must use the secondary index structure.

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
   var
      tb1, tb2 Table
   endVar


   tb1.attach("Customer.db")
   tb1.enumRefIntStruct("CustRef.db")
   tb1.enumFieldStruct("CustFlds.db")
   tb1.enumIndexStruct("CustIdx.db")


   try
      tb2 = CREATE "NewCust.db"
                     struct "CustFlds.db"
                     refIntStruct "CustRef.db"
                     indexStruct "CustIdx.db"
                  ENDCREATE
   onFail
      errorShow()
   endTry


endMethod
```

enumSecStruct method

Lists a table's security information.

Syntax

```
1. enumSecStruct ( const tableName String ) Logical
2. enumSecStruct ( inMem TCursor ) Logical
```

Description

enumSecStruct lists the security information (access rights) of a Table variable. Syntax 1 creates a Corel Paradox table; Syntax 2 stores the information in a TCursor variable.

Syntax 1 creates the Corel Paradox table specified in *tableName*. For dBASE tables, this method lists the structure of the indexes associated with the table by the **usesIndexes** method. If *tableName* is open, this method fails. If *tableName* already exists, this method overwrites it without asking for confirmation. You can also include an alias or path in *tableName*. If an alias or path is not specified, Corel Paradox creates the table in the working directory. You can supply *tableName* to the **secStruct** option in a **create** statement to borrow that table's field structure (including primary keys and validity checks) for use in the new table.

In Syntax 2, the structure information is stored in the TCursor variable *inMem* that you pass as an argument. Syntax 2 results in faster performance because the information is stored in system memory.

The following table displays the structure of the table in Syntax 1 or the TCursor in Syntax 2:

Field name	Type	Description
infoHeader	A1	Specifies whether the record is a header for (and the data it contains is shared by) subsequent consecutive records that have a value of N in this field
iSecNum	N	Specifies the number to identify security description (first description = 1)
eprvTable	N	Specifies the table privilege value
eprvTableSym	A10	Specifies the table privilege name
iFamRights	N	Specifies the family rights value
iFamRightsSym	A10	Specifies the family rights name
szPassword	A31	Specifies the password
fldNum	N	Specifies the ordinal position of field in table
aprvFld	N	Specifies the field privilege value
aprvFldSym	A10	Specifies the field privilege name

Example

{button ,AL(`OPAL_TYPE_TABLE;OPAL_METH_TBENUMFIELDNAMES;OPAL_METH_TBENUMFIELDNAMESINDEX;OPAL_METH_TBENUMFIELDSTRUCT;OPAL_METH_TBENUMINDEXSTRUCT;OPAL_METH_TBENUMREFINTSTRUCT;',0,"Defaultoverview",)} Related Topics

enumSecStruct example

The following example creates a new table based on the security information that is associated with the *Secrets* table. The code uses **enumSecStruct** to write security information to the *SecInfo* table which is then used to create the *MySecrts* table.

```
; getSecrets::pushButton
method pushButton(var eventInfo Event)
var
  tb1, tb2 Table
endVar


tb1.attach("Secrets.db")
tb1.enumSecStruct("SecInfo.db")


tb2 = CREATE "MySecrts.db"
        LIKE "Secrets.db"
        SECSTRUCT "SecInfo.db"
      ENDCREATE


endMethod
```

Privilege values and names for enumSecStruct

The following table lists numeric values and symbolic names for table and field privileges.

Value	Name	Description
0	None	Specifies no privileges
1	ReadOnly	Specifies a read-only field or table
3	Modify	Specifies a read and modify field or table
7	Insert	Specifies insert + all of the above privileges (table only)
15	InsDel	Specifies delete + all of the above privileges (table only)
31	Full	Specifies full rights (table only)
255	Unknown	Specifies privileges unknown

Family rights values and names for enumSecStruct

The following table lists numeric values and symbolic names for family rights.

Value	Name	Description
0	NoFamRights	Specifies no family rights
1	FormRights	Specifies the right to change forms only
2	RptRights	Specifies the right to change reports only
4	ValRights	Specifies the right to change val checks only
8	SetRights	Specifies the right to change image settings
15	AllFamRights	Specifies all of the above

familyRights method

Tests a user's ability to create or modify objects in a table's family.

Syntax

```
familyRights ( const rights String) Logical
```

Description

familyRights determines whether you can create or modify objects in a table's family. This method returns True if you have rights to the type of object specified in *rights*; otherwise, it returns False. *rights* is a single-letter string F (form), R (report), S (image settings), or V (validity checks) that indicates the object type to which you may have rights. This method preserves the functionality required by Corel Paradox 3.5 tables but does not apply to tables created in versions of Corel Paradox after 3.5.

DOS

If you are a DOS PAL programmer, you can use this procedure to operate on tables by specifying the table name, rather than using a variable.

Syntax

```
familyRights( const tableName String, rights AnyType ) Logical
```

Example

{button ,AL(`OPAL_TYPE_TABLE;OPAL_METH_TBTBRI;',0,"Defaultoverview",)} Related Topics

familyRights example

The following example determines whether you have F rights to CUSTOMER.DB.

```
; showFRights::pushButton
method pushButton(var eventInfo Event)
var
  custTB Table
endVar


custTB.attach("Orders.db")
if custTB.isTable() then
  msgInfo("Rights", "Form Rights: " +
          String(custTB.familyRights("F")))
  ;displays True if you have Form rights to Orders.db
else
  msgStop("Error", "Can't find Orders.db.")
endIf


endMethod
```

fieldName method/procedure

Returns the name of a table's field, given a field number.

Syntax

```
fieldName ( const fieldNum SmallInt ) String
```

Description

fieldName returns the name of the field specified in *fieldNum.* If *fieldNum* is greater than the number of fields in the table, **fieldName** returns an empty string.

DOS

If you are a DOS PAL programmer, you can use this procedure to operate on tables by specifying the table name, rather than using a variable.

Syntax

```
fieldName ( const tableName String, const fieldNum SmallInt ) String
```

Example

{button ,AL(`OPAL_TYPE_TABLE;OPAL_METH_TBFNO;',0,"Defaultoverview",)} Related Topics

fieldName example

The following example uses **fieldName** to display the name of field number two in the *Answer* table. This code is attached to the built-in **pushButton** method of a button.

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
  tbl Table
  fldName, tblName String
  fldNum SmallInt
endVar
tblName = "Answer.db"
fldNum = 2


tbl.attach(tblName)
if isTable(tbl) then
  fldName = tbl.fieldName(fldNum)   ; store name of field 2 in fldName
  msgInfo("The name of field " + String(fldNum) + " is:", fldName)
else
  msgStop("Sorry", "Can't find " + tblName + " table.")
endIf


endMethod
```

fieldNo method/procedure

Returns the position of a field in a table.

Syntax

```
fieldNo ( const fieldName String ) SmallInt
```

Description

fieldNo returns the position of the field specified by *fieldName*, or 0 if *fieldName* is not found. Fields are numbered from left to right, beginning with 1.

DOS

If you are a DOS PAL programmer, you can use this procedure to operate on tables by specifying the table name, rather than using a variable.

Syntax

```
fieldNo ( const tableName String, const fieldName String ) SmallInt
```

 **Example**

{button ,AL(`OPAL_TYPE_TABLE;OPAL_METH_TBFNAM;',0,"Defaultoverview",)} **Related Topics**

fieldNo example

The following example displays the field number of the Date field in the Orders table:

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
  ord Table
  fldNo SmallInt
endVar


ord.attach("Orders.db")
fldNo = ord.fieldNo("Date")


if fldNo = 0 then
  msgInfo("Orders table", "Date is not a field in this table.")
else
  msgInfo("Orders table", "Date is field number " + String(fldNo))
endIf


endMethod
```

fieldType method/procedure

Returns the data type of a field in a table.

Syntax

```
1. fieldType ( const fieldName String ) String
2. fieldType ( const fieldNum SmallInt ) String
```

Description

fieldType returns the data type of a field. If the specified field is not found, this method returns "unknown." The following tables list the possible return values for Corel Paradox and dBASE tables:

Corel Paradox Field Type	Return Value
Alpha	ALPHA
Autoincrement	AUTOINCREMENT
BCD	BCD
Binary	BINARY
Bytes	BYTES
Date	DATE
Formatted Memo	FMTMEMO
Graphic	GRAPHIC
Logical	LOGICAL
Long Integer	LONG
Memo	MEMO
Money	MONEY
Number	NUMBER
OLE	OLE
Short	SHORT
Time	TIME
Timestamp	TIMESTAMP

dBASE Field Type	Return Value
BINARY	BINARY
CHARACTER	CHARACTER
DATE	DATE
FLOAT	FLOAT
LOGICAL	LOGICAL
MEMO	MEMO
NUMBER	NUMERIC
OLE	OLE

DOS

If you are a DOS PAL programmer, you can use this procedure to operate on tables by specifying the table name, rather than using a variable.

Syntax

```
1. fieldType ( const tableName String, const fieldName String ) String
2. fieldType ( const tableName String, const fieldNum SmallInt ) String
```

Example

{button ,AL(`OPAL_TYPE_TABLE;OPAL_METH_TBFNO;',0,"Defaultoverview",)} Related Topics

fieldType example

The following example uses a dynamic array to store the data type of each field in the *BioLife* table and displays the contents of the dynamic array in a dialog box.

```
; showFldTypes::pushButton
method pushButton(var eventInfo Event)
var
  tblVar Table
  i SmallInt
  fldTypes DynArray[] AnyType
  tblName String
endVar
tblName = "BioLife.db"


if isTable(tblName) then
  tblVar.attach(tblName)
    ; This FOR loop loads the DynArray with BioLife.db field types.
  for i from 1 to tblVar.nFields()
    fldTypes[tblVar.fieldName(i)] = tblVar.fieldtype(i)
  endFor
    ; Now show the contents of the DynArray.
  fldTypes.view(tblName + " field types")
else
  msgStop("Sorry", "Can't find " + tblName + " table.")
endIf
endMethod
```

getGenFilter method

Retrieves the filter criteria that is associated with a Table variable.

Syntax

```
1. getGenFilter ( criteria DynArray[ ] AnyType ) Logical

2. getGenFilter ( criteria Array[ ] AnyType [ , fieldName Array[ ] AnyType ] ) Logical

3. getGenFilter ( criteria String ) Logical
```

Description

getGenFilter retrieves the filter criteria that is associated with a Table variable. This method assigns values to a dynamic array (DynArray) variable in Syntax 1, or to two Array variables that you declare and include as arguments in Syntax 2.

In Syntax 1, the DynArray *criteria* lists fields and filtering conditions as follows: the index is the field name or number (depending on how it was set), and the item is the corresponding filter expression.

In Syntax 2, the Array *criteria* lists filtering conditions, and the optional Array *fieldName* lists corresponding field names. If you omit *fieldName*, conditions apply to fields in the order they appear in the *criteria* array (the first condition applies to the first field in the table, the second condition applies to the second field, and so on).

If the arrays used in Syntax 2 are resizeable, this method sets the array size to equal the number of fields in the underlying table. If fixed-size arrays are used, this method stores as many criteria as possible, beginning with criteria field 1. If there are more array items than fields, the remaining items are empty. If there are more fields than items, this method fills the array.

In Syntax 3, filter criteria is assigned to a String variable *criteria* that you must declare and pass as an argument.

Examples

{button ,AL(`OPAL_TYPE_TABLE;OPAL_METH_TCSETGENFILTER;OPAL_METH_TCDROPGENFILTER;OPAL_METH_TCSETRANGE;OPAL_METH_UIGETGENFILTER;OPAL_INFO_TBUSINGRANGESANDFILTERS;',0,"Defaultoverview",)} Related Topics

getGenFilter method examples

Example1 Using a **pushButton** method

Example2 Attaching a custom method

getGenFilter example 1

In the following example, the **pushButton** method for a button named *btnShowFilter* uses **getGenFilter** to fill a DynArray named *dyn* with a table's filter criteria. The code then determines whether the current criteria filters the State/Prov field with a value of CA, and resets the filter if necessary.

```
;btnShowFilter :: pushButton
method pushButton(var eventInfo Event)
   var
      custTb         Table
      dyn            DynArray[] AnyType
      keysAr         Array[] AnyType
      stFilterFld,
      stCriteria     String
   endVar


   stFilterFld = "State/Prov"
   stCriteria  = "CA"
   custTb.attach("Customer")


   custTb.getGenFilter(dyn)   ; Get filter information.


   dyn.getKeys(keysAr)
   if keysAr.contains(stFilterFld) then
         if dyn[stFilterFld] = stCriteria then
               return               ; Filter is set correctly.
         endIf
   else
         dyn.empty()             ; Set filter criteria correctly.
         dyn[stFilterFld] = stCriteria
         custTb.setGenFilter(dyn)
   endIf
endMethod
```

getGenFilter example 2

In the following example, a form contains a custom method named *cmGetOrders*. This custom method is used by a button named *btnViewOrders* to set a filter and return the number of records in the filter. The following code is attached to the form:

```
;Form :: cmGetOrders
method cmGetOrders(var tbl Table) Number
   var
      dynCurrent   DynArray[] AnyType
      dynNew      DynArray[] AnyType
   endVar


   dynNew["Ship Via"] = "UPS"      ;Set filter criteria.
   dynNew["Total Invoice"] = "> 10000"
   tbl.getGenFilter(dynCurrent)   ;Get the current criteria.

   if dynCurrent <> dynNew then   ;If current criteria is not
      tbl.setGenFilter(dynNew)      ;the same as new criteria,
   endIf                     ;then set new criteria.


   return(tbl.cCount("Order No"))   ;Return number of orders.
endMethod
```

The following code is attached to the button. It associates a Table variable with a table and calls the custom method attached to the form to operate on the data.

```
;btnViewOrders :: pushButton
method pushButton(var eventInfo Event)
   var
      tbl   Table
   endVar


   tbl.attach("ORD_JUN.DB")
   view(cmGetOrders(tbl), "UPS orders over $10,000 in June")


   tbl.attach("ORD_JUL.DB")
   view(cmGetOrders(tbl), "UPS orders over $10,000 in July")
endMethod
```

getRange method

Retrieves the values that specify a range for a Table variable.

Syntax

```
getRange ( var rangeVals Array[ ] String ) Logical
```

Description

getRange retrieves the values that specify a range for a Table variable. This method assigns values to an Array variable that you declare and include as an argument. The following table displays the array values and the corresponding range criteria:

Number of array items	Range specification
No items (empty array)	Specifies no range criteria is associated with the Table variable
One item	Specifies a value for an exact match on the first field of the index
Two items	Specifies a range for the first field of the index
Three items	The first item specifies an exact match for the first field of the index; items 2 and 3 specify a range for the second field of the index.
More than three items	For an array of size *n*, specifies exact matches on the first *n*-2 fields of the index. The last two array items specify a range for the *n*-1 field of the index

If the array is resizeable, this method sets the array size to equal the number of fields in the underlying table. If fixed-size arrays are used, this method stores as many criteria as it can, starting with criteria field 1. If there are more array items than fields, the remaining items are left empty; if there are more fields than items, this method fills the array and then stops.

Example

{button ,AL(`OPAL_TYPE_TABLE;OPAL_METH_UISETRANGE;OPAL_METH_UIGETGENFILTER;OPAL_METH_TBGETRANGE;OPAL_METH_TCGETRANGE;OPAL_INFO_TBUSINGRANGESANDFILTERS;',0,"Defaultoverview",)} **Related Topics**

getRange example

In the following example, **getRange** is used on a Table variable *tbl* to test if the current range criteria is the same as the new range criteria. If it is not, then the new range is set using **setRange**.

```
;btnSetRange :: pushButton
method pushButton(var eventInfo Event)
   var
      arGet         Array[2] Anytype
      arSet         Array[2] Anytype
   endVar


   arSet[1] = "A"
   arSet[2] = "B"


   ;The following assumes a Table variable
   ;is declared and used elsewhere.


   tbl.getRange(arGet)      ;Get the current range.
   if arGet <> arSet then   ;Compare current range with new.
      tbl.setRange(arSet)      ;Show records starting with A.
   endIf
endMethod
```

index keyword

Creates an index on a the specified fields of a table.

Syntax

```
1. index
       [ maintained ] tableDesc on fieldID
    endIndex
2. index tableDesc
       [ maintained ]              (Corel Paradox)
       [ primary ]                 (Corel Paradox)
       [ caseInsensitive ]         (Corel Paradox)
       [ descending ]              (Corel Paradox and dBASE)
       [ unique ]                  (dBASE)
       on
         { fieldDesc [ , fieldDesc ] [ to indexName ]
           |
         { keyExp
           to ndxFileName|tag tagName [ of mdxFileName ]
             |
           for condition } }
    endIndex
```

Description

index generates a primary or secondary index on the specified fields of a table. Corel Paradox uses the index to accelerate queries and searches that access those fields.

For Corel Paradox tables, the keywords **maintained**, **primary**, and **caseInsensitive** are available. The **primary** keyword specifies a primary index (key), which is required to create any secondary indexes. If the table has a primary index and you create another one, the new index replaces the original. A primary index must be declared on one or more consecutive fields, beginning with the first field in the table. Memo fields, formatted memo fields, OLE fields, and Graphic fields cannot be indexed.

Secondary indexes can be either maintained (created using the **maintained** keyword) or non-maintained. Corel Paradox updates a maintained index as records are added, deleted, or changed. A non-maintained index is only updated when in use. If you use the **maintained** keyword for Corel Paradox tables and specify a non-keyed table to index, **index** fails. For dBASE tables, all opened index files are automatically maintained.

The **caseInsensitive** keyword causes an index to ignore case. A primary index must be case-sensitive. For Corel Paradox tables, a case-sensitive, maintained index on a single field must have the same name as that field. A case-*in*sensitive, maintained index on a single field must *not* have the same name as that field.

The **on** clause specifies which fields to index and two forms: one for Corel Paradox tables, and one for dBASE tables.

For Corel Paradox tables, use

on *fieldDesc* [, *fieldDesc*] to *indexName*

(*fieldDesc* specifies one or more field names or field numbers, and *indexName* specifies the index file. Other methods use this name to refer to the index.)

For dBASE tables, use

keyExp to *ndxFileName*|tag *tagName* [of *mdxFileName*]

(which lets you choose between an .NDX file or a tag in an .MDX file. If *mdxFileName* is omitted, the default .MDX filename is the same as the table. A dBASE table can only be indexed on one field or expression)

In multi-user applications, **index** places a full lock on the table while it is being indexed. If the table has already been locked by another user or application, the command is retried throughout the retry period. If the lock cannot be obtained by the end of the period, **index** fails. You can use the **lock** method to determine whether you can lock the table *before* you use the **index** command.

It can be convenient to develop your applications without worrying about indexes and introduce them where

appropriate to speed up queries and searches.

The index command fails if

- too many indexes already exist (maximum of 255 for a single table)
- an index being defined is already in use

index is not a method, so dot notation is inappropriate. Instead, you create an index structure to specify how to index the table.

For more information on indexes, see About keys and indexes in tables in the User's Guide Help.

Examples

{button ,AL(`OPAL_TYPE_TABLE;OPAL_METH_TBCREA;OPAL_METH_TBCREATEINDEX;OPAL_METH_TBENUMINDEXSTRUCT;OPAL_METH_TBSIND;OPAL_METH_TBUSIND;',0,"Defaultoverview",)} **Related Topics**

index keyword examples

Example1 Building a primary index

Example2 Building a secondary index

index example 1

The following example builds a primary index for a Corel Paradox table named CUSTOMER.DB. If the Customer table can not be found, or cannot be locked, this code aborts the **index** operation. If the table is indexed, the code enumerates indexed fields to an array and displays the array's contents in a dialog box.

```
; newCustKeys::pushButton
method pushButton(var eventInfo Event)
var
  tblToIndex String
  tblVar Table
  indexedFlds Array[] String
endVar
tblToIndex = "Customer.db"


if isTable(tblToIndex) then
  tblVar.attach(tblToIndex)
  if not tblVar.lock("Full") then
    msgStop("Stop!", "Can't lock " + tblToIndex + " table.")
    return
  endIf
  INDEX tblVar              ; create new primary index for Customer.db
    PRIMARY
    ON "Customer No", "Name", "Street"
  ENDINDEX


    ; now display Customer's keyed fields in a dialog box
  tblVar.enumFieldNamesInIndex(indexedFlds)
  indexedFlds.view("Primary key fields for " + tblToIndex)


else
  msgStop("Stop!", "Can't find " + tblToIndex + " table.")
endIf


endMethod
```

index example 2

The following example builds a maintained secondary index named *CityState* for the Corel Paradox table, CUSTOMER.DB. If successful, this code enumerates the indexed field names to an array and displays them in a dialog box:

```
; cityStateIndex::pushButton
method pushButton(var eventInfo Event)
var
  tblToIndex String
  tblVar Table
  indexedFlds Array[] String
  tv TableView
endVar
tblToIndex = "Customer.db"


if isTable(tblToIndex) then
  tblVar.attach(tblToIndex)
  if not tblVar.lock("Full") then
    msgStop("Stop!", "Can't lock " + tblToIndex + " table.")
    return
  endIf


  INDEX tblVar              ; create secondary index for Customer.db
    MAINTAINED              ; maintain index incrementally
    ON "City", "State/Prov" ; index on these two fields
    TO "CityState"          ; name the index "CityState"
  ENDINDEX


    ; now display Customer's keyed fields in a dialog box
  tblVar.enumFieldNamesInIndex("CityState", indexedFlds)
  indexedFlds.view("Fields in the CityState index")


else
  msgStop("Stop!", "Can't find " + tblToIndex + " table.")
endIf


endMethod
```

isAssigned method

Reports whether a Table variable has an assigned value.

Syntax

```
isAssigned ( ) Logical
```

Description

isAssigned returns True if a Table variable has an assigned value; otherwise, it returns False. You can assign a value to a Table variable using **create** or **attach**.

Note

- Even if **isAssigned** returns True, the table may not exist. For example, the following code displays True in a dialog box:

```
var tb Table endVar
tb.attach("zxcv.qw")                  ; attach to some nonsense filename
msgInfo("Assigned?", tb.isAssigned()) ; displays True
displays True in the dialog box.
```

Example

{button ,AL(`OPAL_TYPE_TABLE;OPAL_METH_TBATT;OPAL_METH_TBISTAB;',0,"Defaultoverview",)}

Related Topics

isAssigned example

The following example determines whether the Table variable is assigned before attaching to a table. The following code goes in the Var window for the *thisForm* form:

```
; thisForm::var
var
  tblVar Table
endVar
```

The following code is attached to the **pushButton** method for the *thisButton* button. If *tblVar* is not already assigned, it is attached to the *Orders* table.

```
; thisButton::pushButton
method pushButton(var eventInfo Event)


if NOT tblVar.isAssigned() then
  tblVar.attach("Orders.db")
else
  msgStop("Error", "Can't attach tblVar to Orders.db")
endIf


endMethod
```

isEmpty method/procedure

Reports whether a table contains any records.

Syntax

```
isEmpty ( ) Logical
```

Description

isEmpty returns True if there are no records in a table; otherwise, it returns False.

DOS

If you are a DOS PAL programmer, you can use this procedure to operate on tables by specifying the table name, rather than using a variable.

Syntax

```
isEmpty ( const tableName String ) Logical
```

Example

{button ,AL(`OPAL_TYPE_TABLE;OPAL_METH_TBEMPT;OPAL_METH_TBISTAB;',0,"Defaultoverview",)}

Related Topics

isEmpty example

In the following example, the **pushButton** method for the *rptRecNo* button displays the number of records in the *Orders* table. If *Orders* is empty, this code informs the user:

```
; rptRecNo::pushButton
method pushButton(var eventInfo Event)
var
  tblVar Table
  tblName String
endVar
tblName = "Orders.db"


if isTable(tblName) then
  tblVar.attach(tblName)
  if tblVar.isEmpty() then     ; if Orders.db table is empty
    msgStop("Hey!", tblName + " table is empty!")
  else
    msgInfo(tblName + " table has", String(tblVar.nRecords()) + " records")
  endIf
else
  msgStop("Sorry", "Can't open " + tblName + " table.")
endIf
endMethod
```

isEncrypted method/procedure

Reports whether a table is password-protected.

Syntax

```
isEncrypted ( [ const tableName String ] ) Logical
```

Description

isEncrypted returns True if a table is password-protected; otherwise, it returns False. A TCursor can't be opened on an encrypted table until the password is presented interactively or using the Session type method **addPassword**. To determine whether a user has access rights to the table use **tableRights** .

DOS

If you are a DOS PAL programmer, you can use this procedure to operate on tables by specifying the table name, rather than using a variable.

Syntax

```
isEncrypted ( const tableName String ) Logical
```

Example

{button ,AL(`OPAL_TYPE_TABLE;OPAL_METH_TBPROT;OPAL_METH_TBTBRI;OPAL_METH_SSAPASS;OPAL_METH_SSRMPAS;',0,"Defaultoverview",)} Related Topics

isEncrypted example

The following example uses **isEncrypted** to determine whether the *Secrets* table is password-protected ; if it is, the user must enter a password.

```
method pushButton(var eventInfo Event)
   const
      kTbName = "Secrets"
   endConst

   var
      tbSecret Table
      tvSecret TableView
   endvar

   tbSecret.attach(kTbName)

   ; If the table is encrypted, prompt the
   ; user for the password.

   if tbSecret.isEncrypted() then
      menuAction(MenuFileTablePasswords)
   endIf

   if not tvSecret.open(kTbName) then
      errorShow("Could not open " + kTbName)
   endIf

endMethod
```

isShared method/procedure

Reports whether a table is currently shared with another user on the network.

Syntax

```
isShared ( ) Logical
```

Description

isShared returns True if a table is being shared by another user on a network; otherwise, it returns False. **isShared** does not report whether a table is being shared with another session.

DOS

If you are a DOS PAL programmer, you can use this procedure to operate on tables by specifying the table name, rather than using a variable.

Syntax

```
isShared ( const tableName String ) Logical
```

Example
{button ,AL(`OPAL_TYPE_TABLE;OPAL_METH_TBTBRI;',0,"Defaultoverview",)} **Related Topics**

isShared example

In the following example, a Table variable is attached to the Customer table. This code uses **setExclusive** to give the user exclusive rights to *Customer* then uses **isShared** to demonstrate the effect that **setExclusive** has on tables in a multi-user environment:

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
  tblVar Table
  tblName String
endVar
tblName = "Customer.db"


tblVar.attach(tblName)


tblVar.setExclusive(True)   ; give user exclusive rights to Customer.db
if tblVar.isShared() then   ; this is never True!
                            ; exclusive tables can't be shared
  msgStop("", "This message will never appear!")
else
  msgInfo("Multi-user Status", tblName + " is not shared.")
endIf


endMethod
```

isTable method/procedure

Reports whether a table exists in a database.

Syntax

```
isTable ( ) Logical
```

Description

isTable returns True if the specified Table variable represents a table that can be opened; otherwise, it returns False.

DOS

If you are a DOS PAL programmer, you can use this procedure to operate on tables by specifying the table name, rather than using a variable.

Syntax

```
isTable ( const tableName String ) Logical
```

Example
{button ,AL(`OPAL_TYPE_TABLE;OPAL_METH_TBISASSIGNED;',0,"Defaultoverview",)} **Related Topics**

isTable example

The following example uses **isTable** to determine whether the *Customer* table exists before doing anything with the table. If *Customer* exists in the default database, this code stores *Customer* field names in an array and displays the contents of the array in a dialog box:

```
; showCustFlds::pushButton
method pushButton(var eventInfo Event)
var
  tblVar Table
  tblName String
  fldNames Array[] AnyType
endVar
tblName = "Customer.db"


tblVar.attach(tblName)
if isTable(tblVar) then
  tblVar.enumFieldNames(fldNames)
  fldNames.view(tblName + " fields")
else
  msgStop("Stop!", "Can't find " + tblName + " table.")
endIf


endMethod
```

lock method

Locks a specified table.

Syntax

```
lock ( const lockType String ) Logical
```

Description

lock locks a specified table. The *lockType* argument is one of the following String values, listed in order of decreasing strength and increasing concurrency:

String value	Description
Full	The current session has exclusive access to the table. Cannot be used with dBASE tables.
Write	The current session can write to and read from the table. No other session can place a write lock or a read lock on the table.
Read	The current session can read from the table. No other session can place a write lock, full lock, or exclusive lock on the table.

If successful, **lock** returns True; otherwise, it returns False.

Example

{button ,AL(`OPAL_TYPE_TABLE;OPAL_METH_TBUNLOCK;OPAL_METH_SSLOCK;OPAL_METH_SSUNLOCK;',0,"Defaultoverview",)} Related Topics

lock example

The following example attaches a Table variable to *Customer*, places an exclusive lock on the table and uses **reIndex** to rebuild the *Phone_Zip* index. When the index is rebuilt, this code unlocks *Customer* so other network users can gain access to the table:

```
; reindexCust::pushButton
method pushButton(var eventInfo Event)
var
  tblVar Table
  pdoxTbl String
endVar
pdoxTbl = "Customer.db"


if isTable(pdoxTbl) then
  tblVar.attach(pdoxTbl)
  if tblVar.lock("Exclusive") then    ; Try to lock the table.
    tblVar.reIndex("Phone_Zip")       ; Rebuild Phone_Zip index.
    tblVar.unLock("Exclusive")        ; Unlock the table.
  else
    msgStop("Sorry", "Can't lock " + pdoxTbl + " table.")
  endIf
else
  msgStop("Sorry", "Can't find " + pdoxTbl + " table.")
endIf
endMethod
```

nFields method/procedure

Returns the number of fields in a table.

Syntax

```
nFields ( ) LongInt
```

Description

nFields returns the number of fields in a table.

DOS

If you are a DOS PAL programmer, you can use this procedure to operate on tables by specifying the table name, rather than using a variable.

Syntax

```
nFields ( const tableName String ) LongInt
```

Example

{button ,AL(`OPAL_TYPE_TABLE;OPAL_METH_TBNKEY;OPAL_METH_TBNREC;',0,"Defaultoverview",)}

Related Topics

nFields example

In the following example, the **pushButton** method for *thisButton* displays the number of fields in the *BioLife* table:

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
  tblVar Table
endVar


tblVar.attach("BioLife.db")
msgInfo("BioLife", "BioLife has " +
        String(tblVar.nFields(), " fields."))


endMethod
```

nKeyFields method/procedure

Returns the number of fields in the primary index for a table.

Syntax

```
nKeyFields ( ) LongInt
```

Description

nKeyFields returns the number of fields in the primary index for a table. Use **getIndexName** (TCursor type) to retrieve the index's name.

For more information on indexes, see About keys and indexes in tables in the User's Guide Help.

DOS

If you are a DOS PAL programmer, you can use this procedure to operate on tables by specifying the table name, rather than using a variable.

Syntax

```
nKeyFields ( const tableName String ) LongInt
```

Example

{button ,AL(`OPAL_TYPE_TABLE;OPAL_METH_TCGETINDEXNAME;OPAL_METH_TBNFLD;',0,"Defaultoverview",)} Related Topics

nKeyFields example

The following example returns the number of primary key fields in a Corel Paradox table (ORDERS.DB). This code also returns the number of primary key fields in the LastName tag of the SCORES.MDX index for a dBASE table (SCORES.DBF):

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
  pdoxTbl, dBaseTbl Table
  nkf LongInt
endVar


pdoxTbl.attach("Orders.db")
nkf = pdoxTbl.nKeyFields()        ; number of key fields in the primary index
msgInfo("Orders", "Orders.db has " + String(nkf) + " key fields.")



dBaseTbl.attach("Scores.dbf")
dBaseTbl.setIndex("Scores", "LastName")
nkf = dBaseTbl.nKeyFields() ; key fields in LastName tag
msgInfo("Scores.dbf", "LastName tag has "
                      + String(nkf) + " key fields.")

endMethod
```

nRecords method/procedure

Returns the number of records in a table.

Syntax

```
nRecords ( ) LongInt
```

Description

nRecords returns the number of records in the table associated with a Table variable.

If you call **nRecords** after setting a filter, the return value does not represent the number of records in the filtered set. To retrieve that information, use **cCount**. If you call nRecords after setting a range, the return value represents the number of records in the set defined by the range.

nRecords counts deleted records in dBASE tables if **showDeleted** is turned on. If **showDeleted** is turned off, deleted records are not counted.

DOS

If you are a DOS PAL programmer, you can use this procedure to operate on tables by specifying the table name, rather than using a variable.

Syntax

```
nRecords ( const tableName String ) LongInt
```

Example

Related Topics

nRecords example

The following example prompts the user for confirmation before deleting all records from the *Scratch* table. If the user does not confirm the action, this code uses **nRecords** to indicate how many records exist in SCRATCH.DB:

```
; tblEmpty::pushButton
method pushButton(var eventInfo Event)
var
  tblName String
  tblVar Table
endVar
tblName = "Scratch.db"


if isTable(tblName) then
  tblVar.attach(tblName)
  if msgYesNoCancel("Confirm", "Empty " + tblName + " table?") = "Yes" then
    tblVar.empty()
    message("All " + tblName + " records have been deleted.")
  else
    message(tblname + " has " + String(tblVar.nRecords()) + " records.")
  endIf
else
  msgInfo("Error", "Can't find " + tblName + " table.")
endIf
endMethod
```

protect method/procedure

Assigns an owner password to a table.

Syntax

```
protect ( const password String ) Logical
```

Description

protect assigns an owner password to a table. The password cannot exceed 31 characters. A password-protected table cannot be accessed without presenting the password specified in *password*. If the table already has a password, **protect** fails.

Once a table is protected, you can use the **addPassword** method to present the password, and the **removePassword** method to withdraw the password. *password* is case-sensitive (e.g., a table protected with Sesame won't open for SESAME).

Do not confuse **protect** with **lock**: **protect** encrypts tables, while **lock** controls simultaneous access to tables.

DOS

If you are a DOS PAL programmer, you can use this procedure to operate on tables by specifying the table name, rather than using a variable.

Syntax

```
protect ( const tableName String, const password String ) Logical
```

Example
{button ,AL(`OPAL_TYPE_TABLE;OPAL_METH_TBISEN;OPAL_METH_SSAPASS;OPAL_METH_SSRMPAS;',0,"Defaultoverview",)} **Related Topics**

protect example

In the following example, the **pushButton** method for *protectSecrets* password-protects the *Secrets* table in the default database:

```
; protectSecrets::pushButton
method pushButton(var eventInfo Event)
var
  secretData Table
endVar


secretData.attach("Secrets.db")
if not secretData.isEncrypted() then
  secretData.protect("Get007") ; Password-protect table with "Get007"
endIf


endMethod
```

reIndex method

Rebuilds an index or index tag that is not automatically maintained.

Syntax

1. (Corel Paradox tables) **reIndex (** const ***indexName*** String **)** Logical
2. (dBASE tables) **reIndex (** const ***indexName*** String [const ***tagName*** String] **)** Logical

Description

reIndex rebuilds an index or index tag that is not automatically maintained. In a Corel Paradox table, use *indexName* to specify an index. In a dBASE table, use *indexName* to specify an .NDX file, or *indexName* and *tagName* to specify an index tag in an .MDX file. **reIndex** requires exclusive access to the table.

For more information on indexes, see About keys and indexes in tables in the User's Guide Help.

Example

{button ,AL(`OPAL_TYPE_TABLE;OPAL_METH_TBREINDAL;',0,"Defaultoverview",)} Related Topics

reIndex example

The following example attaches a Table variable to a Corel Paradox table named Customer, places an exclusive lock on the table and uses **reIndex** to rebuild the *Phone_Zip* index:

```
; reindexCust::pushButton
method pushButton(var eventInfo Event)
var
  tblVar Table
  pdoxTbl String
endVar
pdoxTbl = "Customer.db"


tblVar.attach(pdoxTbl)
if tblVar.lock("Exclusive") then  ; Try to lock the table.
  tblVar.reIndex("Phone_Zip")     ; Rebuild Phone_Zip index.
  tblVar.unLock("Exclusive")      ; Unlock the table.
else
  msgStop("Sorry", "Can't lock " + pdoxTbl + " table.")
endIf


endMethod
```

reIndexAll method

Rebuilds all index files associated with a table.

Syntax

```
reIndexAll ( ) Logical
```

Description

reIndexAll rebuilds all index files associated with a table. This method requires exclusive rights to rebuild a maintained index and a write lock to rebuild a non-maintained index.

For more information on indexes, see About keys and indexes in tables in the User's Guide Help.

Example

{button ,AL(`OPAL_TYPE_TABLE;OPAL_METH_TBREIND;',0,"Defaultoverview",)} Related Topics

reIndexAll example

In the following example, the **pushButton** method for a button attempts to place an exclusive lock on the *Customer* table. If **lock** is successful, this code rebuilds all indexes for the *Customer* table and unlocks the table:

```
; reindexAllCust::pushButton
method pushButton(var eventInfo Event)
var
  tblVar Table
  pdoxTbl String
endVar
pdoxTbl = "Customer.db"


tblVar.attach(pdoxTbl)
if tblVar.lock("Exclusive") then    ; attempt to lock Customer.db
  tblVar.reIndexAll()               ; rebuild all Customer.db indexes
  tblVar.unLock("Exclusive")        ; unlock the table
else
  msgStop("Sorry", "Can't lock " + pdoxTbl + " table.")
endIf


endMethod
```

rename method/procedure

Renames a table.

Syntax

```
rename ( const destTableName String ) Logical
```

Description

rename changes a table's name to the name specified by *destTableName*. If the table named by *destTableName* already exists, an error results.

Throughout the retry period, this method attempts to place a full lock on the table. If the lock cannot be placed, an error results.

DOS

If you are a DOS PAL programmer, you can use this procedure to operate on tables by specifying the table name, rather than using a variable.

Syntax

```
rename ( const tableName String, const destTableName String ) Logical
```

Example

{button ,AL(`OPAL_TYPE_TABLE;OPAL_METH_TBCOPY;',0,"Defaultoverview",)} Related Topics

rename example

The following example renames CUSTOMER.DB to OLDCUST. If *OldCust* already exists, this code allows you to abort the operation:

```
; renameCust::pushButton
method pushButton(var eventInfo Event)
var
  tblVar Table
  oldName, newName String
endVar


oldName = "Customer.db"
newName = "OldCust.db"


tblVar.attach(oldName)
if tblVar.isTable() then
  if isTable(newName) then
    if msgQuestion("Confirm", newName + " exists. Overwrite it?") <> "Yes" then
      message("Operation canceled.")
      return
    endIf
  endIf
  tblVar.rename(newName)
  message(oldName + " renamed to " + newName)
else
  msgStop("Stop!", "Can't find " + oldName + " table.")
endIf

endMethod
```

restructure method

Restructures a table.

Syntax

```
restructure( const createSpec DynArray[ ] AnyType ) Logical
```

Description

restructure allows you to modify a table's structure under program control. You can add, delete, or modify the fields in your table, create indexes, change referential integrity relationships, and so on. **restructure** also allows you to perform other operations that are available when restructuring a table (e.g., packing the table).

restructure uses a dynamic array named *createSpec,* which contains the information on the changes to make to the table.

To specify the kind of change to be made, the field, index, referential integrity, and security structure tables, if specified, must include an ID field. This ID field is named slightly differently in each structure file, however, all end with id. Use this ID field to specify the type of operation to perform. The RestructureOperations constants *crModify*, *crAdd*, and *crDrop* are provided for each operation.

restructure returns True if successful, and False if the restructure operation fails, or if a Keyviol or Problems table is generated. It also fails if it cannot obtain a full lock on the table.

The following clauses specify table attributes in *createSpec* and are optional. They can appear in any order within the dynamic array:

saveAs specifies a new name for the restructured table, leaving the original table unchanged. By default, the restructured table is saved with the same name, which overwrites the original table.

Keyviol specifies the table to which any records causing a key violation are saved.

Problems specifies the table to which any problem records are saved. If data is lost during the restructure, the problem records are placed in the problems table, and the operation that caused the problem is not performed on those records.

fieldStruct specifies the name of the table from which you can borrow field structure information. Use **enumFieldStruct** to generate the field structure table before executing **restructure**. The *_Invariant Field Id* field of the field structure table contains the original field number of the table to be restructured. Use the *_Invariant Field Id* field to specify the change to be made to the table. To add a field, insert a new record in the field structure table describing the new field and place *crAdd* in the *_Invariant Field Id* field. To delete an existing field, remove the record from the field structure table.

indexStruct specifies the name of the table from which you can borrow index structure information. Use **enumIndexStruct** to generate the index structure table (or create it manually). Use the *iIndexId* field of the index structure table to specify the change to be made to the table. To modify an index, use *crModify* in the *iIndexId* field. You can modify an index name by dropping (*crDrop*), and adding (*crAdd*) another record with the changed name.

refIntStruct specifies the name of the table from which you can borrow referential integrity structure information. Use **enumRefIntStruct** to generate the referential integrity structure table (or create it manually). Use the *iRefId* field of the referential integrity structure table to specify the change to be made to the table. Use crModify to modify existing values, *crAdd* to add, or *crDrop* to delete.

secStruct specifies the table from which you can borrow security structure information. Use **enumSecStruct** to generate the security structure table name (or create it manually). Use the *iSecId* field in the security structure table to specify the change to be made to the table. Use crModify to modify existing values, *crAdd* to add, or *crDrop* to delete.

pack specifies whether to pack the table. Valid values are True or False. For more information, see **compact** (Table type).

versionLevel specifies the table version level. See **create** for a listing of version numbers for Corel Paradox and dBASE tables.

languageDriver specifies the language driver name. For a list of language drivers for Corel Paradox tables, see Language drivers for Corel Paradox tables. For dBASE tables, see Language drivers for dBASE tables.

blockSize specifies the size of data blocks used to store information in the table, in kilobytes. (A kilobyte is 1,024 bytes.) Valid block sizes depend on the file format of the table. For Corel Paradox versions 4.5 or earlier, 1K through 4K are valid. For versions 5.0 and later, 1K through 4K, 8K, 16K, and 32K are valid.

warnings specifies whether warnings encountered during the restructure operation are displayed. Valid values are True or False. If **warnings** is set to True, warnings can be placed on the error stack for examination.

If **errorTrapOnWarnings** is set to True, the first warning generated terminates the restructure operation.

Example

{button ,AL(`OPAL_TYPE_TABLE;OPAL_METH_TBCREA;OPAL_METH_TBENUMFIELDSTRUCT;OPAL_METH_TB

ENUMINDEXSTRUCT;OPAL_METH_TBENUMREFINTSTRUCT;OPAL_METH_TBENUMSECSTRUCT;',0,"Defaultoverview",)} **Related Topics**

restructure example

The following example appears in a script window and modifies the Customer table by changing a field name. This code uses **enumFieldStruct** to create the field structure information, updates the information, copies the updated information to a dynamic array and passes the dynamic array to the restructure method:

```
method run(var eventInfo Event)
var
     tbl         Table
     tcFlds      TCursor
     dynNewStru  DynArray[] Anytype
endvar

tbl.attach( "Customer.db" )
tbl.enumFieldStruct( "field_struct.db" )

tcFlds.open("field_struct.db" )
tcFlds.edit()

scan tcFlds :
   if tcFlds."Field Name" = "Name" then
      tcFlds."Field Name" = "Company Name"
      quitLoop
   endif
endscan
tcFlds.endEdit()
tcFlds.close()

dynNewStru["FIELDSTRUCT"] = "field_struct.db"
tbl.restructure( dynNewStru )
endmethod
```

setExclusive method

Specifies whether to grant the user exclusive rights to a table when it is opened.

Syntax

```
setExclusive ( [ const yesNo Logical ] )
```

Description

setExclusive specifies whether to open a table with shared or exclusive rights. This method does not place locks on the table an exclusive lock is placed on the table only when it is opened. Exclusive locks are more powerful than full locks.

By default, tables are opened in shared mode. Optional argument *yesNo* specifies whether to set exclusive rights. A value of Yes requests exclusive rights so that no other user can read or write to the table; a value of No allows the table to be opened in shared mode. By default, *yesNo* is set to Yes.

Example

{button ,AL(`OPAL_TYPE_TABLE;OPAL_METH_TBATT;OPAL_METH_TBSIND;OPAL_METH_TBSETREADONLY;', 0,"Defaultoverview",)} Related Topics

setExclusive example

The following example demonstrates how **setExclusive** affects access rights to a table. This code defines a Table variable for the *Customer* table and calls **setExclusive** so *Customer* is opened exclusively. Then, a TCursor is opened for *Customer*. If the TCursor is successfully opened, it has exclusive rights to the table and **lockStatus** is called to indicate that an exclusive lock has been placed on *Customer*.

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
  tblVar Table
  tc     TCursor
endvar


tblVar.attach("Customer.db")
if tblVar.isTable() then
  ; set exclusive rights for the Table variable
  tblVar.setExclusive()


  ; attempt to open a TCursor on Customer.db
  ; if successful, tc has exclusive rights to Customer.db
  if tc.open(tblVar) then

    ; if tc.open was successful, this message indicates
    ; that tc has 1 exclusive lock on Customer.db
    msgInfo("Lock Status", tc.lockStatus("Exclusive"))


  else
    ; else open failed
    msgInfo("Status", "Can't open Customer.db")
  endIf


else
  msgInfo("Status", "Can't find Customer.db table.")
endIf


if tc.isAssigned() then    ; if the TCursor was opened
  tc.close()               ; close tc

now Customer.db is not
                          ; locked and can be opened by another user
endIf


endMethod
```

setGenFilter method

Specifies conditions for including records in a TCursor opened on a Table variable.

Syntax

```
1. setGenFilter ( criteria DynArray[ ] AnyType ) Logical
2. setGenFilter ( criteria Array[ ] AnyType [ , fieldId Array[ ] AnyType ] ) Logical
```

Description

setGenFilter specifies conditions for including records in a TCursor opened on a Table variable. Records that meet the specified conditions are included in the TCursor. Records that don't meet the criteria are filtered out, creating a restricted view of the table. **setGenFilter** must be executed before opening a table with a TCursor.

In Syntax 1, a dynamic array (DynArray) named *criteria* specifies fields and filtering conditions. The index is the field name or number, and the item is the filter expression.

The following code specifies criteria based on the values of three fields:

```
criteriaDA[1]      = "Widget"            ; The value of the first field
                                         ; in the table is Widget.


criteriaDA["Size"] = "> 4"               ; The value of the field named
                                         ; Size is greater than 4.


criteriaDA["Cost"] = ">= 10.95, < 22.50" ; The value of the field named
                                         ; Cost is greater than or
                                         ; equal to 10.95  and less
                                         ; than 22.50.
```

If the DynArray is empty, all existing filter criteria are removed.

In Syntax 2, an Array named *criteria* specifies filtering conditions, and an optional Array named *fieldId* specifies field names and numbers. If you omit *fieldID*, conditions are applied to fields in the order they appear in the *criteria* array (the first condition applies to the first field, the second condition applies to the second field, and so on). The following example specifies the same criteria as the example for Syntax 1:

```
criteriaAR[1] = "Widget"
criteriaAR[2] = "> 4"
criteriaAR[3] = ">= 10.95, < 22.50"
fieldAR[1] = 1
fieldAR[2] = "Size"
fieldAR[3] = "Cost"
```

If the Array is empty, all existing filter criteria are removed.

Examples

{button ,AL(`OPAL_TYPE_TABLE;OPAL_METH_TBGETGENFILTER;OPAL_METH_TBDROPGENFILTER;OPAL_METH_TCSETGENFILTER;OPAL_METH_UISETGENFILTER;OPAL_INFO_TBUSINGRANGESANDFILTERS;',0,"Defaultoverview",)} Related Topics

setGenFilter method examples

Example1 Attaching a table variable with a built-in **run** method

Example2 Attaching a table variable with a **pushButton** method

setGenFilter example 1

In the following example, the built-in **run** method for a script attaches a Table variable to the *Customer* table and sets filter criteria on the *State* field to equal CA:

```
;Script :: run
method run(var eventInfo Event)
   var
      tb         Table
      dyn      DynArray[] AnyType
   endVar


   dyn["State/Prov"] = "CA"


   tb.attach("CUSTOMER.DB")
   tb.setGenFilter(dyn)


endMethod
```

setGenFilter example 2

In the following example, a form contains a button named *btnBalanceStatus*. The **pushButton** method for *btnBalanceStatus* attaches a Table variable to the *Orders* table and sets filter criteria that displays only those records with a positive balance. **cCount** then retrieves the number of records, **cAverage** retrieves the average balance due, and **cSum** retrieves the total balance due. Finally, a dialog box displays the values.

```
;btnBalanceStatus
method pushButton(var eventInfo Event)
   var
      tbl         Table
      dyn         DynArray[] AnyType
      s1,
      s2,
      s3            String
   endVar


   tbl.attach("ORDERS")
   Dyn["Balance Due"] = "> 0"
   tbl.setGenFilter(Dyn)


   s1 = string(tbl.cCount("Balance Due"))
   s2 = string(tbl.cAverage("Balance Due"))
   s3 = string(tbl.cSum("Balance Due"))


   msgInfo("Outstanding balances", "There are " + s1 + " orders with an average balance due of
" + s2 + ", totaling " + s3 + ".")
endMethod
```

setIndex method

Specifies an index for a table.

Syntax

1. (Corel Paradox tables) **setIndex (** const ***indexName*** String **)** Logical
2. (dBASE tables) **setIndex (** const ***indexName*** String [**,** const ***tagName*** String] **)** Logical

Description

setIndex specifies an index to use when a table is opened.

In a Corel Paradox table, use *indexName* to specify an index. In a dBASE table, you can use *indexName* to specify an .NDX file, or *indexName* and *tagName* to specify an index tag in an .MDX file.

For more information on indexes, see About keys and indexes in tables in the User's Guide Help.

Example

{button ,AL(`OPAL_TYPE_TABLE;OPAL_METH_TBSETRANGE;',0,"Defaultoverview",)} Related Topics

setIndex example

The following example assumes that the Corel Paradox Customer table has a secondary index named CityState. The following code specifies CityState with **setIndex** to set up for a call to **setRange**. When the filter is set for *Customer*, this code loads a DynArray with information from the filtered table then displays the DynArray's contents in a dialog box:

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
  custTbl Table
  tc TCursor
  dy DynArray[] Anytype
endVar


custTbl.attach("Customer.db")
if isTable(custTbl) then


  ; now use the secondary index named CityState
  custTbl.setIndex("CityState")


  ; filter out everything but St. Thomas
  custTbl.setRange("St. Thomas", "St. Thomas")


  ; open a TCursor for the filtered Customer table
  if tc.open(custTbl) then


    ; scan the table and load the DynArray with
    ; company names (Name) and phone numbers
    scan tc:
      dy[tc.Name] = tc.Phone
    endScan
    ; display contents of the DynArray
    dy.view("St. Thomas Phone Numbers")


  else
    msgStop("Error", "Can't open TCursor.")
  endIf

else
  msgStop("Error", "Can't find Customer.db")
endIf
endMethod
```

setRange method

Specifies a range of records to associate with a Table variable. This method enhances the functionality of **setFilter**, which it replaces in this version. Code that calls **setFilter** continues to execute as before.

Syntax

```
1. setRange ( [ const exactMatchVal AnyType] * [ , const minVal AnyType, const maxVal AnyType ] ) Logical

2. setRange ( rangeVals Array[ ] AnyType ) Logical
```

Description

setRange specifies conditions for associating a contiguous range of records with a Table variable. Records that meet the conditions are included when the table is opened. **setRange** compares the criteria you specify with values in the corresponding fields of a table's index. If the table is not indexed, this method fails. If you call **setRange** without any arguments, the range criteria is reset to include the entire table.

Syntax 1 allows you to set a range based on the value of the first field of the index by specifying values in *minVal* and *maxVal*. For example, the following statement examines values in the first field of the index of each record:

```
tableVar.setRange(14, 88)
```

If a value is less than 14 or greater than 88, that record is filtered out. To specify an exact match on the first field of the index, assign the same value to *minVal* and *maxVal*. For example, the following statement filters out all values except 55:

```
tableVar.setRange(55, 55)
```

To set a range based on the values of more than one field, specify exact matches *except* for the last one in the list. For example, the following statement looks for exact matches on Corel and Corel Paradox (assuming they are the first fields in the index), and values ranging from 100 to 500 (inclusive) for the third field:

```
tableVar.setRange("Corel", "Corel Paradox", 100, 500)
```

Syntax 2 allows you to pass an array of values to specify the range criteria, as listed in the following table.

Number of Array Items	Range Specification
No items (empty array)	Resets range criteria to include the entire table
One item	Specifies a value for an exact match on the first field of the index
Two items	Specifies a range for the first field of the index
Three items	The first item specifies an exact match for the first field of the index; items 2 and 3 specify a range for the second field of the index.
More than three items	For an array of size *n*, specify exact matches on the first *n*-2 fields of the index. The last two array items specify a range for the *n*-1 field of the index.

Examples

Related Topics

setRange method examples

Example1 Calculating totals in a field

Example2 Calling **setRange** with a criteria array of more than three items

setRange example 1

The following example assumes that Lineitem's key field is Order No. and that you want to know the total for order number 1005. The following code attaches a Table variable to the *Lineitem* table, limits the range of records to those with 1005 in the first field of the primary index and uses **cSum** to calculate the total for order number 1005:

```
; getDetailSum::pushButton
method pushButton(var eventInfo Event)
var
  tblVar Table
  tblName String
endVar
tblName = "LineItem.db"
tblVar.attach(tblName)


  ; this limits TCursor's view to records that have
  ; 1005 in the first field of the primary index
tblVar.setRange(1005, 1005)


  ; now display the total for Order No. 1005
msgInfo("Total for Order 1005", tblVar.cSum("Total"))


endMethod
```

setRange example 2

The following example calls **setRange** with a criteria array that contains more than three items. The following code sets a range to include orders from a person with a specific first name, middle initial, and last name, and an order quantity ranging from 100 to 500 items. The code then counts the number of records in this range and displays the value in a dialog box. This example assumes that the *PartsOrd* table is indexed on the FirstName, MiddleInitial, LastName, and Qty fields:

```
; setQtyRange::pushButton
method pushButton(var eventInfo Event)
   var
      tbPartsOrd   Table
      arRangeInfo   Array[5] AnyType
      nuCount      Number
   endVar


   arRangeInfo[1] = "Frank"      ; FirstName (exact match)
   arRangeInfo[2] = "P."         ; MiddleInitial (exact match)
   arRangeInfo[3] = "Corel"    ; LastName (exact match)
   arRangeInfo[4] = 100          ; Minimum qty value
   arRangeInfo[5] = 500          ; Maximum qty value


   if tbPartsOrd.attach("PartsOrd") then
         tbPartsOrd.setRange(arRangeInfo)
         nuCount = tbPartsOrd.cCount(1)
         nuCount.view("Number of big orders by Frank P. Corel:")
   else
         errorShow("Can't open the table.")
   endIf
endMethod
```

setReadOnly method

Specifies whether to grant the user read-only rights to a table when it is opened.

Syntax

```
setReadOnly ( [ const yesNo Logical ] )
```

Description

setReadOnly specifies whether to grant the user read-only rights to a table when it is opened. This method fails if the table has been locked by another user or if the table is open.

Optional argument *yesNo* specifies whether to set read-only rights: a value of Yes grants read-only rights, a value of No allows full rights to the table. By default, *yesNo* is set to Yes.

Example

{button ,AL(`OPAL_TYPE_TABLE;OPAL_METH_TBSETEXCLUSIVE;',0,"Defaultoverview",)} Related Topics

setReadOnly example

The following example attaches a Table variable to the Orders table, calls **setReadOnly** to limit rights and opens a TCursor for Orders:

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
  tblVar Table
  tc TCursor
endVar


errorTrapOnWarnings()
tblVar.attach("Orders.db")  ; attach Table var to Orders.db
tblVar.setReadOnly()        ; set Table to read-only
tc.open(tblVar)             ; open a TCursor for Orders.db
tc.edit()


endMethod
```

showDeleted method

Specifies whether to display deleted records in a dBASE table.

Syntax

```
showDeleted ( [ const yesNo Logical ] ) Logical
```

Description

showDeleted specifies whether to display deleted records in a dBASE table. Records deleted from a dBASE table aren't immediately removed. Instead, they are flagged for deletion and removed later. **showDeleted** is relevant only for dBASE tables.

Optional argument *yesNo* specifies whether to display deleted records (a value of Yes) or hide deleted records (a value of No). By default, *yesNo* is set to Yes. If you don't call this method before using the Table variable associated with the table, deleted records are not displayed.

Example

{button ,AL(`OPAL_TYPE_TABLE;OPAL_METH_TBCOM;OPAL_METH_TBDELETE;',0,"Defaultoverview",)}

Related Topics

showDeleted example

In the following example, the **pushButton** method attached to the *showDeletedRecs* button displays a Table variable's deleted records:

```
; showDeletedRecs::pushButton
method pushButton(var eventInfo Event)
var
  tblVar Table
endVar


tblVar.attach("Orders.dbf")
if isTable(tblVar) then


  ; show deleted records in Orders.dbf
  tblVar.showDeleted(Yes)


  ; display sum of deleted and undeleted records
  msgInfo("Total # of Records", tblVar.nRecords())
else
  msgStop("Error", "Can't find Orders table.")
endIf


endMethod
```

sort keyword

Sorts a table.

Syntax

```
sort sourceTable [ on fieldNameList [ D ] ] [ to destTable ]    endSort
```

Description

sort sorts the table specified in *sourceTable* .

sourceTable can be a Table, TCursor, or String type. *destTable* can be a Table or String type. However, you can't sort a TCursor onto itself.

If you include the optional **on** clause, the table is sorted on the first field specified in *fieldNameList*. Each subsequent field settles ties in the preceding fields. An optional **D** after a field name specifies a sort in descending order. If you omit the **on** clause, records are sorted in ascending order, moving from left to right across the fields.

If you include the optional **to** clause, the sort result written to the table specified by *destTable*. If that table already exists, it is overwritten without asking for confirmation. If you omit the **to** clause, the sorted records are returned to *sourceTable* (this fails if the table is open). You must specify the **to** clause if the source table is keyed.

sort automatically places a full lock on the tables being sorted if the result is written to the same table. Otherwise, a write lock is required for the source table and a full lock for the target table.

sort is not a method, so dot notation is inappropriate. Instead, you create a structure to specify how to sort the table.

Example

{button ,AL(`OPAL_TYPE_TABLE;OPAL_METH_TBINDE;OPAL_METH_TCSORTTO;',0,"Defaultoverview",)}

Related Topics

sort example

The following example sorts *Customer* on the Last Name and First Name fields, and displays the results in the *CustSort* table:

```
; sortCustTable::pushButtton
method pushButton(var eventInfo Event)
var
  custTbl Table
  tv TableView
endVar


custTbl.attach("Customer.db")


sort custTbl
  on "Country" D, "Name" D     ; sort in descending order
  to "CustSort.db"
endSort


tv.open("CustSort.db")                 ; open the sorted table


endMethod
```

subtract method/procedure

Subtracts the records in one table from another table.

Syntax

```
1. subtract ( const destTableName String ) Logical
2. subtract ( const destTableName Table ) Logical
```

Description

subtract determines whether records that reside in the source table also reside in *destTableName*. If matching records are found, **subtract** deletes them from *destTableName* without asking for confirmation.

If *destTableName* is keyed, **subtract** deletes the records with keys that match the values of key fields in the source table. If *destTableName* is not keyed, **subtract** deletes the records that match any record in the source table. Whether tables are keyed or not, this method considers only fields that *could* be keyed (based on data type, not position). For example, numeric fields are considered, but formatted memos are not. This method requires read/write access to both tables.

If the target table is not keyed, this operation can be time-consuming.

Throughout the retry period, this method attempts to place a full lock on both tables. If locks cannot be placed, an error results.

DOS

If you are a DOS PAL programmer, you can use this procedure to operate on tables by specifying the table name, rather than using a variable.

Syntax

```
1. subtract ( const sourceTableName String, const destTableName String ) Logical
2. subtract ( const sourceTableName String, const destTableName Table ) Logical
```

Example

{button ,AL(`OPAL_TYPE_TABLE;OPAL_METH_TBADD;',0,"Defaultoverview",)} Related Topics

subtract example

The following example subtracts records found in the *Inserted* table (in the private directory) from the Customer table:

```
; subtractCust::pushButton
method pushButton(var eventInfo Event)
var
  insTbl, CustTbl Table
  fs FileSystem
  tblName String
endVar
tblName = privDir() + "\\Inserted.db"


insTbl.attach(tblName)
if insTbl.isTable() then
  insTbl.subtract(custTbl)    ; remove from custTbl matching records in insTbl
else
  msgInfo("Sorry", "Can't find " + tblName + " table.")
endIf


endMethod
```

tableRights method/procedure

Specifies whether the user has the right to perform table operations.

Syntax

```
tableRights ( const rights String ) Logical
```

Description

tableRights specifies whether the user has the right to perform table operations. The following table describes *rights*:

Value	Description
ReadOnly	Specifies the right to read from the table without making changes
Modify	Specifies the right to enter or change data
Insert	Specifies the right to add new records
InsDel	Specifies the right to add and delete records
Full or All	Specifies the right to perform all of the above operations

This method returns True if the user has the specified rights; otherwise, it returns False.

DOS

If you are a DOS PAL programmer, you can use this procedure to operate on tables by specifying the table name, rather than using a variable.

Syntax

```
tableRights ( const tableName String, const rights String )
```

Example

{button ,AL(`OPAL_TYPE_TABLE;OPAL_METH_TBFAMR;',0,"Defaultoverview",)} Related Topics

tableRights example

The following example reports whether the user has All rights to the Orders table:

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
  myRights Logical
  ordTbl   Table
endVar


ordTbl.attach("Orders.db")
if ordTbl.isTable() then
  myRights = ordTbl.tableRights("All")

  ; this displays True if you have All rights to Orders.db
  msgInfo("All Rights?", myRights)

else
  message("Can't find Orders table.")
endIf
endMethod
```

type method

Returns a table's type.

Syntax

```
type ( ) String
```

Description

type returns the string value COREL PARADOX or DBASE to specify the table's type.

Example

{button ,AL(`OPAL_TYPE_TABLE;OPAL_METH_TBATT;',0,"Defaultoverview",)} **Related Topics**

type example

The following example removes deleted records from the *Orders* table if **type** returns DBASE. If **type** returns Corel Paradox, a message is displayed:

```
; compactButton::pushButton
method pushButton(var eventInfo Event)
var
  tblVar Table
endVar
tblVar.attach("Orders")
if tblVar.type() = "DBASE" then
  tblVar.compact()
else
  msgStop("Stop!", "Orders is a " + tblVar.type() + " table.")
endIf


endMethod
```

unAttach method

Ends the association between a Table variable and a table description.

Syntax

```
unAttach ( ) Logical
```

Description

unAttach ends the association (created using **attach** or **create**) between a Table variable and a table description. You don't have to end the association between a Table variable and a table to attach the same variable to another table. **unAttach** is automatically called when a Table variable is assigned to a different table.

Example

{button ,AL(`OPAL_TYPE_TABLE;OPAL_METH_TBATT;',0,"Defaultoverview",)} Related Topics

unAttach example

In the following example, a Table variable is used to summarize sales information from two different tables. When the Table variable (*tableVar*) is no longer needed, this code calls **unAttach** to end the association between *tableVar* and the table:

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
  tableVar Table
  q1, q2   Number
  msg      String
endVar


tableVar.attach("q1_sales.db")  ; attach to q1_sales table
q1 = tableVar.cSum("Amount")    ; get a summary


tableVar.attach("q2_sales.db")  ; no need to unattach
q2 = tableVar.cSum("Amount")    ; get summary from q2_sales


tableVar.unAttach()             ; we don't need tableVar anymore
                                ; so end the association to q2_sales


switch
   case q2 < q1 : msg = "Sales are down."
   case q2 = q1 : msg = "Sales are flat."
   case q2 > q1 : msg = "Sales are up."
endSwitch

msgInfo("Sales", msg)


endMethod
```

unlock method

Unlocks a specified table.

Syntax

```
unlock ( const lockType String ) Logical
```

Description

unlock removes locks that are explicitly placed on the table associated with a Table variable. *lockType* is one of the following String values, listed in order of decreasing strength and increasing concurrency:

String value	Description
Full	The current session has exclusive access to the table. No other session can open the table. Cannot be used with dBASE tables.
Write	The current session can write to and read from the table. No other session can place a write lock or a read lock on the table.
Read	The current session can read from the table. No other session can place a write lock, full lock, or exclusive lock on the table.

unlock removes locks that have been explicitly placed by a particular user or application using **lock**. **unlock** has no effect on locks placed automatically by Corel Paradox. To ensure maximum concurrent availability of tables unlock a table that has been explicitly locked as soon as the lock is no longer needed. If you lock a table twice, you must unlock it twice. You can use **lockStatus** (defined for the TCursor and UIObject types) to determine how many explicit locks you have placed on a table. If you try to unlock a table that isn't locked or cannot be unlocked, **unlock** returns False .

If successful, this method returns True; otherwise, it returns False.

Example

{button ,AL(`OPAL_TYPE_TABLE;OPAL_METH_TBLOCK;OPAL_METH_TCLSTA;OPAL_METH_UILOCKSTATUS;', 0,"Defaultoverview",)} **Related Topics**

unlock example

In the following example, the **pushButton** method for *updateCust* runs a query from an existing file and adds records from the *Answer* table to the *Customer* table. This code attempts to place a write lock on the *Customer* table before adding records to it. If the lock is placed, this code adds *Answer* records and uses **unlock** to unlock *Customer*:

```
; updateCust::pushButton
method pushButton(var eventInfo Event)
var
  newCust Query
  ansTbl Table
  destTbl String
endVar
destTbl = "Customer.db"

newCust.readFromFile("getCust.qbe")

if newCust.executeQBE() then           ; If the query succeeds,
  ansTbl.attach(":PRIV:Answer.db")
  if destTbl.lock("Write") then        ; try to write lock the table.
    ansTbl.add(destTbl)                ; Add records from Answer.db.
    destTbl.unLock("Write")            ; Unlock the table.
  else
    msgStop("Stop", "Can't write lock " + destTbl + " table.")
  endIf
else
  msgStop("Stop!", "Query failed.")
endIf

endMethod
```

unProtect method/procedure

Permanently removes an owner password from a table.

Syntax

```
1. ( Procedure ) unProtect ( const tableName String [ , const Password String ] )
2. ( Method ) unProtect ( [ const password String ] )
```

Description

unProtect permanently removes an owner password from a table. A protected table is encrypted and cannot be accessed without presenting the password that is specified in *password*. If you have already issued the master password for a table, *password* is not necessary.

Example

{button ,AL(`OPAL_TYPE_TABLE;OPAL_METH_TBISEN;OPAL_METH_TBPROT;OPAL_METH_SSAPASS;OPAL_METH_SSRMPAS;',0,"Defaultoverview",)} Related Topics

unProtect example

The following example permanently removes password protection from the *Secrets* table:

```
; decrypt::pushButton
method pushButton(var eventInfo Event)
var
  tblVar Table
  tblName String
endVar


tblName = "Secrets.db"
tblVar.attach(tblName)
if tblVar.isEncrypted() then
  tblVar.unprotect("Get007")   ; permanently remove password
                               ; this assumes Get007 is the master password
endIf


endMethod
```

usesIndexes method

Specifies index files to use and maintain with a dBASE table.

Syntax

```
usesIndexes ( const indexFileName String [ , const indexFileName String ] * Logical
```

Description

usesIndexes specifies one or more index files (.NDX and .MDX) to maintain while you use a dBASE table. This method specifies index files to open when the table is opened. This method is not used to open production files (e.g., the .MDX file with the same name as the table) for a dBASE table. These files are opened automatically.

If any of the specified index files do not exist, this method fails.

For more information on indexes, see About keys and indexes in tables in the User's Guide Help.

Example

{button ,AL(`OPAL_TYPE_TABLE;OPAL_METH_TBREIND;OPAL_METH_TBREINDAL;OPAL_METH_TBSIND;',0,"Defaultoverview",)} Related Topics

usesIndexes example

The following example calls **usesIndexes** to specify two different indexes in the *Orders* table and opens a TCursor for the table:

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
  tblVar Table
  tc     TCursor
endvar


tblVar.attach("Orders.dbf")
if tblVar.isTable() then


  ; specify NameStat and Ord_Name indexes
  tblVar.usesIndexes("NAMESTAT.NDX", "ORD_NAME.NDX")


  ; now attempt to open the table, using the specified indexes
  if tc.open(tblVar) then
    if tc.locate("State", "FL", "Contact", "Simons") then
      msgInfo("Order Date", tc."Order Date")
    else
      msgStop("Error", "Can't find values.")
    endIf
  endIf
else
  msgStop("Error", "Can't find Orders.dbf table.")
endIf
endMethod
```

Using ranges and filters

Although ranges and filters allow you to select a subset of the records in a Table variable, a TCursor, or a UIObject, they operate differently.

A range is based on the fields in an index. When you apply a range to a table, a subset of records that are contiguous and consecutive is created. For this reason, a range gives faster performance than a filter.

A filter offers greater flexibility when selecting fields and specifying criteria. A filter is based on any table field and can use expressions to specify criteria. For example, a filter can select records in which the Quantity field has values of 125, 200, and 350. A range could only specify values ranging from 125 to 350.

For more information on indexes, see About keys and indexes in tables in the User's Guide Help.

{button ,AL(`OPAL_TYPE_TABLE;OPAL_METH_TBSETGENFILTER;OPAL_METH_TCSETGENFILTER;OPAL_METH_UISETGENFILTER;OPAL_METH_TBSETRANGE;OPAL_METH_TCSETRANGE;OPAL_METH_UISETRANGE;',0,"Defaultoverview",)} Related Topics

TableView type

A TableView object displays a table's data in its own window. A TableView object is distinct from a table frame, which is a UIObject placed in a form, and from a TCursor, which is a programmatic construct that points to the data in a table.

If you declare a TableView variable and open a TableView object to that variable, a handle to the TableView window is created. You can refer to the handle in your code to manipulate the TableView object.

TableView methods are a subset of the Form type methods and control the Table window's size, position, and appearance. Although you can start and end Edit mode for a table view, you cannot use ObjectPAL to directly edit the data in a table view. You can use ObjectPAL to manipulate TableView properties in the following areas:

- the TableView object as a whole (e.g., background color, grid style, number of records, and the value of the active record)
- the field-level data in the table (e.g., font, color, and display format (TVData))
- the TableView heading (e.g., font, color, and alignment (TVHeading))

The TableView type includes several derived methods from the Form type.

Methods for the TableView type

Form	**TableView**
bringToTop	**action**
getPosition	**close**
getTitle	**moveToRecord**
hide	**open**
isMaximized	**wait**
isMinimized	
isVisible	
maximize	
minimize	
setPosition	
setTitle	
show	
windowHandle	

Print related ObjectPAL methods and examples

action method

Performs an action command.

Syntax

```
action ( const actionID SmallInt ) Logical
```

Description

action performs the action specified by the constant *actionId*. *actionId* is a constant in one of the following action classes:

- ActionDataCommands
- ActionEditCommands
- ActionFieldCommands
- ActionMoveCommands
- ActionSelectCommands

You can also use **action** to send a user-defined action constant to a built-in **action** method. User-defined action constants are integers that don't interfere with any of ObjectPAL's constants. User-defined constants can be used to signal other parts of an application. For example, assume that the Const window for a form declares a constant named *myAction*. You can use the **id** method to verify the value of each incoming ActionEvent. If the value is equal to *myAction*, you can respond to that action accordingly. By default, Corel Paradox passes the action to the **action** method.

The **action** method is distinct from the built-in **action** method for a TableView or for any form or UIObject. The built-in **action** method for an object responds to an action event; this method causes an ActionEvent.

Example

{button ,AL(`OPAL_TYPE_TABLEVIEW;OPAL_METH_TVOPE;',0,"Defaultoverview",)} Related Topics

action example

The following example opens a TableView for the *Orders* table, moves the cursor to the end of the table, starts Edit mode, and inserts a new record. This code is attached to the **pushButton** method for a button named *startEditInsert*:

```
; startEditInsert::pushButton
method pushButton(var eventInfo Event)
var
  orderTV  TableView
endVar
if orderTV.open("Orders") then
  orderTV.action(DataEnd)           ; move to the end of the table
  orderTV.action(DataBeginEdit)     ; start Edit mode
  orderTV.action(DataInsertRecord)  ; insert a new blank record
  orderTV.wait()                    ; wait until TableView object is closed
  orderTV.close()                   ; close when return
else
  msgStop("Status", "Could not find Orders table.")
endIf
endMethod
```

close method

Closes a table window.

Syntax

```
close ( )
```

Description

close closes a table window. **close** performs the same function as the Close command in the Control menu.

Example

{button ,AL(`OPAL_TYPE_TABLEVIEW;OPAL_METH_TVOPE;',0,"Defaultoverview",)} **Related Topics**

close example

In the following example, a form's **open** method opens a TableView object for the *Customer* table to a global variable named *custTV*. When the form closes, its **close** method closes the *custTV* TableView. This code is attached to the **close** method for the form:

```
; thisForm::close
method close(var eventInfo Event)
if eventInfo.isPreFilter()
  then
    ; code here executes for each object in form
  else
    ; code here executes just for form itself
    custTV.close()     ; close the Customer table that was
                       ; opened by thisForm's open method
endIf
endMethod
```

The following code is attached to the form's Var window:

```
; thisForm::Var
Var
  custTV  TableView   ; global to form, the TableView object is opened by
                      ; form's open method
endVar
```

The following code is attached to the form's **open** method:

```
; thisForm::open
method open(var eventInfo Event)
if eventInfo.isPreFilter()
  then
    ; code here executes for each object in form
  else
    ; code here executes just for form itself
    custTV.open("Customer")  ; open the Customer table view
endIf
endMethod
```

moveToRecord method

Moves to a specific record in a table.

Syntax

```
moveToRecord ( const tc TCursor ) Logical
```

Description

moveToRecord moves to the record pointed to by a TCursor named *tc*. Use the **RecNo** property to accelerate performance in dBASE tables.

Example

{button ,AL(`OPAL_TYPE_TABLEVIEW;OPAL_METH_TVACTION;',0,"Defaultoverview",)} **Related Topics**

moveToRecord example

The following example uses a TCursor to search for a customer named Jones and calls **moveToRecord** to display that record. The following code is attached to a button's built-in **pushButton** method:

```
method pushButton (var eventInfo Event)
var
   custTC TCursor
   custTV TableView
endVar


custTC.open ("customer.db")
custTV.open ("customer.db")


if custTC.locate ("Name", "Ocean Paradise") then
  custTV.moveToRecord (custTC)
else
   msgInfo("Search failed", "Couldn't find Ocean Paradise.")
endIf


endMethod
```

open method

Opens a table window.

Syntax

```
1. open ( const tvName String [ , const windowStyle LongInt ] ) Logical
2. open ( const tvName String, const windowStyle LongInt, const x SmallInt, const y SmallInt,
const w SmallInt, const h SmallInt ) Logical
```

Description

open opens the table specified by *tvName* in a table window. Optional arguments specify (in twips) the location of the upper-left corner of the form (*x* and *y*), the form's width and height (*w* and *h*), and the form's style (*windowStyle*). The *windowStyle* argument is required for Syntax 2. To specify a size and position for the form, use a window style constant (WinStyleDefault).

Example

{button ,AL(`OPAL_TYPE_TABLEVIEW;OPAL_METH_TVCLO;',0,"Defaultoverview",)} **Related Topics**

open example

In the following example, the **pushButton** method for a button named *openWaitOrders* opens the *Orders* table:

```
; openWaitOrders::pushButton
method pushButton(var eventInfo Event)
var
  ordersTV  TableView
endVar
if ordersTV.open("Orders", WinStyleDefault, 100, 100,
                 1440*5, 1440*4) then
  ordersTV.wait()    ; wait for user to close
  ordersTV.close()   ; close Orders table
endIf
endMethod
```

wait method

Suspends a method's execution.

Syntax

```
wait ( )
```

Description

wait suspends a method's execution. Execution resumes when the TableView object is closed. When a TableView object has been called by **wait**, the method suspends execution until the TableView object is closed using the **close** method.

Example

{button ,AL(`OPAL_TYPE_TABLEVIEW;OPAL_METH_TVCLO;OPAL_METH_TVOPE;',0,"Defaultoverview",)}

Related Topics

wait example

See the **open** example.

TCursor type

A TCursor is a pointer to data that is contained in a table. Using TCursors, you can manipulate a table's data without displaying the table. When you edit records in a TCursor, the underlying table is changed. Locks on the table affect the TCursor. A TCursor can point to an entire table or to a subset of the records in a table (e.g., those specified by a restricted view, detail set, filter, or range).

For more information about related objects, see the Table, TableView, and UIObject types.

Some table operations require Corel Paradox to create temporary tables in the private directory.

Methods for the TCursor type

enumSecStruct
enumTableProperties
eot
familyRights
fieldName
fieldNo
fieldRights
fieldSize
fieldType
fieldUnits2
fieldValue
forceRefresh
getGenFilter
getIndexName
getLanguageDriver
getLanguageDriverDesc
getRange
handle
home
initRecord
insertAfterRecord
insertBeforeRecord
insertRecord
instantiateView
isAssigned
isEdit
isEmpty
isEncrypted
isOpenOnUniqueIndex
isRecordDeleted
isShared
isShowDeletedOn
isValid
isView
locate
locateNext
locateNextPattern
locatePattern
locatePrior
locatePriorPattern
lock
lockRecord
lockStatus
moveToRecord
moveToRecNo
nextRecord
nFields
nKeyFields
nRecords

open
postRecord
priorRecord
qLocate
recNo
recordStatus
reIndex
reIndexAll
seqNo
setBatchOff
setBatchOn
setFlyAwayControl
setFieldValue
setGenFilter
setRange
showDeleted
skip
subtract
switchIndex
tableName
tableRights
type
unDeleteRecord
unlock
unlockRecord
updateRecord

 **Print related ObjectPAL methods and examples**

add method

Adds records from one table to another table.

Syntax

```
1. add ( const destTable String [ , const append Logical [ , const update Logical ] ] ) Logical

2. add ( const destTable Table [ , const append Logical [ , const update Logical ] ] ) Logical

3. add ( const destTable TCursor [ , const append Logical [ , const update Logical ] ] )
Logical
```

Description

add adds the records pointed to by a TCursor to the target table specified in *destTable*. If the destination does not exist, this method creates it. The source table and the target table can be the same type or different types; in any case, the tables must have compatible field structures.

When set to True, *append* adds records at the end of a non-indexed target table, or at the appropriate place in an indexed target table. When set to True, *update* compares records in both tables, and where key values match, replaces the data in the target table. When both are set to True, records with matching key values are updated, and others are appended. These arguments are optional, but if you specify *update*, you must also specify *append.* By default, both arguments are True.

```
myTCursor.add("yourTable", False, True) ; specifies update
myTCursor.add("yourTable") ; specifies update and append by default
```

Key violations (including validity check violations) are listed in KEYVIOL.DB in the private directory. If KEYVIOL.DB already exists, **add** overwrites it. If KEYVIOL.DB does not exist, this method creates it.

When tables are keyed, **add** uses the keyed fields to determine which records to update and which to append. If the target table is not keyed and *update* is set to True, **add** fails. **add** respects the limits of restricted views set by **setRange** or **setGenFilter**.

Throughout the retry period this method tries to place write locks on the source and target tables. If either lock cannot be placed, the method fails.

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCCOPY;OPAL_METH_TCSUB;',0,"Defaultoverview",)}

Related Topics

add example

The following example assumes that the *OldCust* and *NewCust* tables exist in the active directory. The following code associates a TCursor with each of the tables, adds *NewCust* records to *OldCust* and adds all records to a table named *MyCust*. If *MyCust* does not exist in the active directory, **add** creates it. This code is attached to a button's **pushButton** method:

```
; getMyCust::pushButton
method pushButton(var eventInfo Event)
var
  TC1, TC2 TCursor
endVar


if TC1.open("oldCust.db") and
   TC2.open("newCust.db") then  ; if both TCursors can be associated

   TC2.add(TC1, True)           ; append oldCust records to newCust

                                ; records now TC1 has
                               ; records from both tables
  TC1.add("myCust.db", True)   ; add TC1 to myCust table


   TC1.close()                 ; close both TCursors
   TC2.close()
else
   msgStop("Stop!", "Could not open one or more tables.")
endIf

endMethod
```

aliasName method

Returns a TCursor's alias.

Syntax

```
aliasName ( ) String
```

Description

aliasName returns a string containing a TCursor's alias. Only TCursors that were opened using an alias returns an alias name. If the TCursor was not opened using an alias, **aliasName** returns an empty string.

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCADD;',0,"Defaultoverview",)} Related Topics

aliasName example

The following example uses **aliasName** to determine the OPEN MODE property value for the open TCursor:

```
method pushButton(var eventInfo Event)
var
   actualPropVal,
   expectedPropVal,
   propertyName,
   tableName       String
   tc              TCursor
endVar


;// initialize variables
propertyName = "OPEN MODE"           ;// SQL alias property name
expectedPropVal = "READ/WRITE"       ;// SQL alias property value
tableName = ":Interbase4:Customer"   ;// SQL table name (includes
                                     ;// the SQL alias name)


if tc.open( tableName ) then
   ;// Get the current property value by specifying the alias name
   ;// using tc.aliasName() and compare with expected value
   actualPropVal = getAliasProperty( tc.aliasName(), propertyName )


   if actualPropVal = expectedPropVal then
      msgInfo("SQL Table Access Mode", actualPropVal)
   else
      ;// try to set to the desired property by specifying the alias
      ;// name using tc.aliasName() and notify the user
      setAliasProperty( tc.aliasName(), propertyName, expectedPropVal )
      msgInfo("SQL Table Access Mode Updated", actualPropVal)
   endif
endIf
endMethod
```

atFirst method

Reports whether the TCursor is pointing to the table's first record.

Syntax

atFirst ( ) Logical

Description

atFirst returns True if the TCursor is pointing to the table's first record; otherwise, it returns False.

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCALAS;OPAL_METH_TCBOT;OPAL_METH_TCEOT;',0,"Defaultoverview",)} **Related Topics**

atFirst example

The following example assumes that a form has a button named *moveToFirst* and a multi-record object bound to ORDERS.DB. The code attached to the **pushButton** method for *moveToFirst* uses **atFirst** to determine whether the TCursor points to the first record. If **atFirst** returns False, this code moves the TCursor to the first record:

```
; moveHome::pushButton
method pushButton(var eventInfo Event)
var
  tc TCursor
endVar


tc.attach(orders)         ; orders is a multi-record object
if not tc.atFirst() then  ; if not at the first record
  tc.home()               ; move to it
  orders.moveToRecord(tc) ; move highlight to first record
else
  msgStop("Currently on record " + String(tc.recNo()),
          "You're already at the top of the list!")
endIf
endMethod
```

atLast method

Reports whether the TCursor is pointing to the table's last record.

Syntax

```
atLast ( ) Logical
```

Description

atLast returns True if the TCursor is pointing to the table's last record; otherwise, it returns False.

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCAFIR;OPAL_METH_TCBOT;OPAL_METH_TCEOT;',0,"Defaultoverview",)} **Related Topics**

atLast example

The following example assumes that a form has a button named *moveToLast* and a multi-record object bound to ORDERS.DB. The code attached to the **pushButton** method for *moveToLast* uses **atLast** to determine whether the TCursor points to the last record. If **atLast** returns False, this code moves the TCursor to the last record:

```
; moveToLast::pushButton
method pushButton(var eventInfo Event)
var
  tc TCursor
endVar


tc.attach(ORDERS)
if not tc.atLast() then   ; if not on the last record
  tc.end()                ; move TCursor to the last record
  orders.moveToRecord(tc) ; move highlight to the last record
else
  msgStop("Currently on record " + String(tc.recNo()),
          "You're already at the last record!")
endIf
endMethod
```

attach method

Associates a TCursor with a specified table.

Syntax

```
1. attach ( const object UIObject ) Logical
2. attach ( const srcTCursor TCursor ) Logical
3. attach ( const tv TableView ) Logical
4. attach ( const srcHandle LongInt ) Logical
```

Description

attach associates a TCursor with a specified table. The data (including filters, indexes, and edit mode) comes from the underlying table. The TCursor retrieves data from committed records only (and not from records being edited or inserted).

Syntax 1 associates a TCursor with the table displayed in a UIObject named *object*.

Syntax 2 associates the TCursor with the table represented by another TCursor, named *srcTCursor*.

Syntax 3 associates the TCursor with the TableView object named *tv*.

Syntax 4 associates the TCursor with the opened cursor handle named *srcHandle.* The TCursor's data comes from the underlying cursor, pointed to by *srcHandle*, which is typically from an external DLL call. **attach** clones the cursor for use in ObjectPAL. The external DLL is responsible for opening and closing the cursor. Explicitly close the TCursor using a **TCursor.close()** in Corel Paradox before closing the cursor in the external DLL.

attach returns True if successful; otherwise, it returns False and adds the following warning to the error stack:

"You have tried to access a document that is not open."

Examples

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCISVA;',0,"Defaultoverview",)} Related Topics

attach method examples

Example 1 Using a **pushButton** method

Example 2 Using code contained in a script window

attach example 1

The following example assumes that a form contains a table frame bound to ORDERS.DB, and another table frame bound to LINEITEM.DB. The *Orders* table has a one-to-many link to *LineItem*. The form also contains a button named *findDetails* . To allow your users to search the entire *LineItem* table. The **pushButton** method for *findDetails* searches for orders that include the current part number.

The following code is attached to the Var window for the *findDetails* button:

```
; findDetails::Var
Var
  lineTC TCursor  ; instance of LINEITEM for searching
endVar
```

The following code is attached to the **open** method for the *findDetails* button. This code associates the *lineTC* TCursor with LINEITEM.DB:

```
; findDetails::open
method open(var eventInfo Event)
   lineTC.open("LineItem.db")
endMethod
```

The following code is attached to the **pushButton** method for *findDetails*:

```
; findDetails::pushButton
method pushButton(var eventInfo Event)
var
   stockNum,
   orderNum   Number
   orderTC      TCursor
endVar
; Get Stock No from current LineItem record.
stockNum = LINEITEM.Stock_No

; LineTC was declared in Var window and opened by open method.
if NOT lineTC.locateNext("Stock No", stockNum) then
  lineTC.locate("Stock No", stockNum)
endIf


orderTC.attach(ORDERS)       ; Attach TCursor to table frame.
orderTC.locate("Order No", lineTC."Order No")
ORDERS.moveToRecord(orderTC) ; Move to CUSTOMER and
                             ; resynchronize with TCursor.
LINEITEM.moveTo()            ; Move TCursor to LINEITEM detail.


; Move TCursor to matching record.
LINEITEM.locate("Stock No", stockNum)
endMethod
```

The following code is attached to the **close** method for *findDetails*:

```
; findDetails::close
method close(var eventInfo Event)
lineTC.close()   ; Close the TCursor to LineItem.
endMethod
```

attach example 2

The following example is contained in a Script window. PDXTEST.DLL contains the openTable(), moveTo(), and closeTable() methods. This code opens a cursor by calling the DLL's openTable() method. A returned handle *hcur*, an ObjectPAL TCursor attaches to the DLL's cursor, and displays the TCursor's active record number. The DLL's moveTo() method is then used to change the cursor's position to record 3. **attach** is called to resynchronize ObjectPAL's TCursor with the DLL's cursor. ObjectPAL's TCursor and the DLL's cursor are then closed.

```
; describe the methods from PDXTEST.DLL that will be called
Uses PDXTEST
   openTable ( tableName CPTR) CLONG [stdcall]
   moveTo    ( pos CLONG ) [stdcall]
   closeTable() [stdcall]
endUses
method run(var eventInfo Event)
var
   tc   tcursor
   hCur LongInt
endvar

; Returns a cursor to the table
hCur = openTable( "aspace.db" )

; Attach to the open cursor, and get the record position.
; (When attaching to the open cursor, Corel Paradox creates a clone of hCur.)
tc.attach( hCur )
view(tc.recNo())

; Move to the 3rd record by calling the moveTo method of the DLL.
; (The DLL's cursor's record position is changed, not ObjectPAL's
; TCursor.)
moveTo( 3 )

; Reattach the cursor to sync to the new cursor position.
tc.attach( hCur )
view(tc.recNo())

; Close the ObjectPAL cloned cursor before closing the handle
; in the DLL
tc.close()

; Close the handles in the DLL. The DLL is responsible for closing
; all handles opened by the DLL.
closeTable()

endMethod
```

attachToKeyViol method

Attaches a TCursor to the original record when a key violation occurs.

Syntax

```
attachToKeyViol ( const oldTC TCursor ) Logical
```

Description

attachToKeyViol attaches a TCursor to the original record after a key violation occurs. Specify the TCursor that points to the record that caused the key violation (the new, unposted record).

This method allows you to compare conflicting records before replacing or discarding changes to an existing record. *oldTC* must already be pointing to the new (yet unposted) record.

 **Example**

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCPOSTRECORD;OPAL_METH_TCUPDATERECORD;',0,"Defaultoverview",)} **Related Topics**

attachToKeyViol example

The following example uses **attachToKeyViol** a key violation occurs. The code declares two TCursors: *keyViolTC* and *originalRecTC*. The code opens *keyViolTC* for the *Orders* table and deliberately inserts a record whose key value conflicts with another record. The code then forces a key violation by posting the new record to the table. If the user chooses to view the existing record, the code calls **attachToKeyViol**, attaches the second TCursor (*originalRecTC*) to the original record, and displays the record in a **view** dialog box. If the user chooses to update the original record with data from the new record, this code calls **updateRecord**:

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
  keyViolTC, originalRecTC TCursor
  rec DynArray[] AnyType
endvar


keyViolTC.open("Orders.db")          ; open TCursor for Orders
keyViolTC.edit()                     ; put TCursor in Edit mode
keyViolTC.insertRecord()             ; insert a new record
keyViolTC."Order No" = 1011          ; 1011 is a duplicate key


; if this attempt to post the new record fails
if NOT keyViolTC.postRecord() then


  ; attach originalRecTC to the existing record
  originalRecTC.attachToKeyViol(keyViolTC)


  ; give user the option to see the existing record
  if msgQuestion("Key Exists!",
     "Do you want to see the existing record?") = "Yes" then


    originalRecTC.copyToArray(rec)   ; copy existing record to rec
    rec.view("Original Record")      ; display rec in a dialog box


  endIf


  ; give user the option to replace the existing record
  if msgQuestion("Confirm Update",
     "Do you want to replace existing record?") = "Yes" then


    ; force the new record to post
    keyViolTC.updateRecord(True)
  else
    message("Original record left intact.")
    sleep(1500)
  endIf
else
  message("Posted order number 1011.")
endIf


endMethod
```

bot method

Determines whether a command attempts to move past the table's first record.

Syntax

bot ( ) Logical

Description

bot returns True if a command attempts to move past the table's first record; otherwise, it returns False. **bot** is reset by the next move operation.

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCAFIR;OPAL_METH_TCALAS;OPAL_METH_TCEND;OPAL_METH_TCEOT;OPAL_METH_TCHOME;',0,"Defaultoverview",)} Related Topics

bot example

The following example moves a TCursor backwards through a table and displays a message. This code is attached to a button's **pushButton** method:

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
  myTable TCursor
endVar
myTable.open("sites.db")
myTable.end()                          ; moves to end of table
while myTable.bot() = False            ; loop until we hit the top
   myTable.priorRecord()               ; move backwards through table
endWhile
msgInfo("The Top", "You're at the beginning.")
msgInfo("At the top?", myTable.bot()) ; displays True
myTable.nextRecord()
msgInfo("At the top?", myTable.bot()) ; displays False
endMethod
```

cancelEdit method

Ends Edit mode without saving changes to the active record.

Syntax

```
cancelEdit ( ) Logical
```

Description

cancelEdit ends Edit mode without saving changes to the active record. Use **cancelEdit** before committing or unlocking the record. Once you move the TCursor, changes to the record are committed.

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCCOED;OPAL_METH_TCENDED;',0,"Defaultoverview",)}

Related Topics

cancelEdit example

The following example is attached to the **pushButton** method for the *changeKey* button. This code associates a TCursor with the *Customer* table and attempts to change a value in a keyed field. If the record cannot be posted (e.g., because of a key violation) an error message is displayed and **cancelEdit** is called to restore the record to the original values and end Edit mode:

```
; changeKey::pushButton
method pushButton(var eventInfo Event)
var
  tc TCursor
  rec Array[] AnyType
endVar


tc.open("Customer.db")
if tc.locate("Customer No", 1231) then
  tc.edit()
  tc."Customer No" = 1221   ; attempt to change key value
  if not tc.endEdit() then  ; if endEdit fails
    errorShow("Can't complete the operation.")
    tc.cancelEdit()         ; restore record and leave edit mode
    message("Record left intact.")
  else
    message("Key value changed.")
  endIf
else
  errorShow("Can't find Customer 1231")
endIf


endMethod
```

cAverage method

Returns the average of values in a column of fields.

Syntax

```
1. cAverage ( const fieldName String ) Number
2. cAverage ( const fieldNum SmallInt ) Number
```

Description

cAverage returns the average of values in the column of fields specified by *fieldName* or *fieldNum*. This method respects the limits of restricted views set by **setRange** or **setGenFilter**. **cAverage** handles blank values as specified in the **blankAsZero** setting for the session.

Throughout the retry period, this method attempts to place a write lock on the table. If a lock cannot be placed, the method fails.

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCCCNT;OPAL_METH_TCCSUM;OPAL_METH_TCCMAX;OPAL_METH_TCCMIN;OPAL_METH_TCCSTD;OPAL_METH_SSBLANKASZERO;',0,"Defaultoverview",)} Related Topics

cAverage example

The following example uses **cAverage** to calculate the average order size in the *Orders* table. This code is attached to the **pushButton** method for the *getAvgSales* button:

```
; getAvgSales::pushButton
method pushButton(var eventInfo Event)
var
  ordTC TCursor
  avgSales Number
endVar


; open TCursor for ORDERS table
ordTC.open("Orders.db")
; store average invoice total in avgSales variable
avgSales = ordTC.cAverage("Total Invoice")
; display avgSales in a dialog
msgInfo("Average Order size", avgSales)


endMethod
```

cCount method

Returns the number of values in a column of fields.

Syntax

```
1. cCount ( const fieldName String ) LongInt
2. cCount ( const fieldNum SmallInt ) LongInt
```

Description

cCount returns the number of values in a column of fields specified by *fieldName* or *fieldNum*. **cCount** works for all field types. If the field is numeric, this method handles blank values as specified in the **blankAsZero** setting for the session. If the field is non-numeric and contains blank fields, **cCount** returns the number of nonblank values in the column of fields.

This method respects the limits of restricted views set by **setRange** or **setGenFilter**.

Throughout the retry period, this method attempts to place a read lock on the table. If a lock cannot be placed, the method fails.

cCount is especially useful for returning the number of entries used by another column function.

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCCAVE;OPAL_METH_TCCSUM;OPAL_METH_TCCMAX;OPAL_METH_TCCMIN;OPAL_METH_TCCSTD;OPAL_METH_SSBLANKASZERO;',0,"Defaultoverview",)} **Related Topics**

cCount example

The following example opens a TCursor for a table and uses **cCount** to display the number of records in the TCursor. This code is attached to the **pushButton** method for the *lineItemInfo* button:

```
; lineItemInfo::pushButton
method pushButton(var eventInfo Event)
var
numbersTC TCursor
avgQty Number
numRecs LongInt
endVar
numbersTC.open("Lineitem.db")
avgQty = numbersTC.cAverage("Qty")
numRecs = numbersTC.cCount(4)        ; assumes Quantity is field 4
msgInfo("Average quantity", "Average quantity: " +
String(avgQty) + " \nbased on " + String(numRecs) + " records.")


endMethod
```

close method

Closes a table.

Syntax

```
close ( ) Logical
```

Description

close closes a TCursor, and makes the TCursor variable unassigned. If the active record cannot be committed, **close** closes the TCursor and discards any changes to the record.

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_ATISAS;OPAL_METH_TCOPEN;',0,"Defaultoverview",)}

Related Topics

close example

The following example opens a TCursor for a table, displays information found in the table's last record and closes the TCursor. This code displays a message indicating whether the TCursor variable remains assigned when the TCursor is closed, and is attached to the built-in **pushButton** method for *thisButton*:

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
  tc TCursor
endVar


tc.open("Orders.db")  ; open TCursor for the Orders table
tc.end()              ; move to the end of the table


; display information in the last record
msgInfo("Last Order", "Order number: " + String(tc."Order No") +
        " \nOrder date: " + String(tc."Sale Date"))


tc.close()                                  ; close tc TCursor
msgInfo("Is tc Assigned?", tc.isAssigned())  ; displays False


endMethod
```

cMax method

Returns the maximum value of a column of fields.

Syntax

```
1. cMax ( const fieldName String ) Number
2. cMax ( const fieldNum SmallInt ) Number
```

Description

cMax returns the maximum value in the column of fields specified by *fieldName* or *fieldNum*. If a field is numeric, this method handles blank values as specified in the **blankAsZero** setting for the session. This method respects the limits of restricted views set by **setRange** or **setGenFilter**.

Throughout the retry period, this method attempts to place a read lock on the table. If a lock cannot be placed, the method fails.

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCCAVE;OPAL_METH_TCCMIN;OPAL_METH_TCCSUM;OPAL_METH_TCCCNT;OPAL_METH_TCCSTD;OPAL_METH_SSBLANKASZERO;',0,"Defaultoverview",)} Related Topics

cMax example

The following example assumes that a form has a button named *getMaxBalance*, and a table frame that is bound to the *Orders* table. The **pushButton** method for *getMaxBalance* associates the table frame with a TCursor and locates the highest balance due in the *Orders* table:

```
; getMaxBalance::pushButton
method pushButton(var eventInfo Event)
var
  ordTC TCursor
endVar


ordTC.attach(ORDERS)   ; ORDERS is a table frame on the form


; now locate the maximum value in the "Balance Due" field
ordTC.locate("Balance Due", ordTC.cMax("Balance Due"))
; synchronize the table frame to the TCursor
ORDERS.moveToRecord(ordTC)


endMethod
```

cMin method

Returns the minimum value in a column of fields.

Syntax

```
1. cMin ( const fieldName String ) Number
2. cMin ( const fieldNum SmallInt ) Number
```

Description

cMin returns the minimum value in the column of fields specified by *fieldName* or *fieldNum*. If the field is numeric, this method handles blank values as specified in the **blankAsZero** setting for the session. This method respects the limits of restricted views set by **setRange** or **setGenFilter**.

Throughout the retry period, this method attempts to place a read lock on the table. If a lock cannot be placed, the method fails.

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCCMAX;OPAL_METH_TCCAVE;OPAL_METH_TCCCNT;OPAL_METH_TCCSTD;OPAL_METH_TCCSUM;OPAL_METH_SSBLANKASZERO;',0,"Defaultoverview",)} Related Topics

cMin example

The following example calculates the minimum values in the ORDERS.DB table:

```
; showMinimums::pushButton
method pushButton(var eventInfo Event)
var
  OrdTC TCursor
  minBalDue, minOrder Number
endVar
OrdTC.open("Orders.db")
minBalDue = ordTC.cMin("Balance Due")  ; get minimum balance due
minOrder  = ordTC.cMin(6) ; assumes "Total Invoice" is field 6


; display results in a dialog box
msgInfo("Minimums", "Minimum balance due: " +
String(minBalDue) + "\n" +
                    "Minimum order : " + String(minOrder))
endMethod
```

cNpv method

Returns the net present value of a column, based on a discount or interest rate.

Syntax

```
1. cNpv ( const fieldName String, const discRate Number ) Number
2. cNpv ( const fieldNum SmallInt, const discRate Number ) Number
```

Description

cNpv returns the net present value of the entries in a column of fields. The net present value calculation is based on the interest or discount rate specified by *discRate. discRate* is a decimal number (e.g., 12 percent is expressed as .12). This method handles blank values as specified in the **blankAsZero** setting for the session.

Throughout the retry period, this method attempts to place a read lock on the table. If a lock cannot be placed, the method fails. This method respects the limits of restricted views set by **setRange** or **setGenFilter**.

This method calculates net present value using the following formula:

cNpv = sum(*p*=1 to *n*) of *Vp*/(1+*i*)*p*

(where *n* = number of periods, *Vp* = cash flow in *p*th period, and *i* = interest rate per period)

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCCAVE;OPAL_METH_TCCSUM;',0,"Defaultoverview",)}

Related Topics

cNpv example

The following example associates a TCursor with the *GoodFund* table, then calculates the net present value for the *Expected Return* field. The net present value is calculated based on a monthly interest rate. This code is attached to the **pushButton** method for the *calcNPV* button:

```
; calcNPV::pushButton
method pushButton(var eventInfo Event)
var
  SavingsTC TCursor
  goodFundNPV, apr Number
endVar
SavingsTC.open("GoodFund.db")  ; associate TCursor with Savings table
apr = .125                     ; annual percentage rate


; now calculate net present value based on monthly interest rate
goodFundNPV = SavingsTC.cNpv("Expected Return", (apr / 12))
msgInfo("Net present value", goodFundNPV)


endMethod
```

compact method

Removes deleted records from a dBASE table.

Syntax

```
compact ( [ const regIndex Logical ] ) Logical
```

Description

compact removes deleted records from a dBASE table. Deleted records are not immediately removed from a dBASE table. Instead, they are flagged as deleted and kept in the table. The optional argument *regIndex* specifies whether to regenerate or update the indexes associated with the table. When *regIndex* is set to True, this method regenerates all indexes associated with the TCursor and frees any unused space in the indexes. If *regIndex* is set to False, indexes are not regenerated. By default, *regIndex* is set to True.

This method fails if any locks have been placed on the table, or if the table is open. If the table has maintained indexes, this method requires exclusive access; otherwise it requires a write lock.

This method returns True if successful; otherwise, it returns False.

The **compact** method defined for the TCursor type does not work with Corel Paradox tables. To pack a Corel Paradox table, use the **compact** method defined for the Table type.

Example

 Related Topics

compact example

The following example removes deleted records from a dBASE table named OLDDATA.DBF. This code is attached the **pushButton** method for the *purgeTable* button:

```
; purgeTable::pushButton
method pushButton(var eventInfo Event)
var
tb Table
tc TCursor
endVar
tb.attach("OldData.dbf")
   tb.setExclusive()            ; Get exclusive rights to the table.

   tc.open(tb)                  ; Associate TCursor with OldData table.

   if tc.compact() then         ; Remove all deleted records.
tc.close()
message("Old records purged.")
else
errorShow()
endIf
endMethod
```

copy method

Copies a table.

Syntax

```
1. copy ( const tableName String ) Logical
2. copy ( const tableName Table ) Logical
```

Description

copy copies a table to the target table named *tableName*. If *tableName* does not exist, **copy** creates it. If *tableName* already exists, **copy** overwrites it without asking for confirmation.

Throughout the retry period, this method attempts to place a write lock on the source table and a full lock on the target table. This method fails if either lock cannot be placed, or if the target table is open.

This method does not respect the limits of restricted views.

For more information, see Copying to a different table type in the User's Guide help.

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCADD;OPAL_METH_TCCREC;',0,"Defaultoverview",)}

Related Topics

copy example

The following example copies the *Customer* table to the *NewCust* table. This code uses the **isTable** method (from the DataBase type) to test whether *NewCust* exists; if it does, the user is prompted to confirm the action before *NewCust* is overwritten:

```
; copyCust::pushButton
method pushButton(var eventInfo Event)
var
  sourceTC TCursor
  destTb Table
endVar
destTb.attach("NewCust.db")
sourceTC.open("Customer.db")


; if NewCust.db exists, ask for confirmation
if isTable(destTb) then
  if msgYesNoCancel("Copy table", "Overwrite Newcust.db?") = "Yes" then


    ; copy Customer.db records to NewCust.db
; If .VAL file contains only RI info, it is not copied.
    sourceTC.copy(destTb)
  endIf
endIf


endMethod
```

copyFromArray method

Copies data from an array to the fields of the active record.

Syntax

```
1. copyFromArray ( const ar Array[ ] AnyType ) Logical
2. copyFromArray ( const ar DynArray[ ] AnyType ) Logical
```

Description

copyFromArray copies the elements of an array or a dynamic array (DynArray) to the record pointed to by a TCursor. The TCursor must be in Edit mode.

Syntax 1 uses an array named *ar*. The first element of the array is copied to the first field, the second element to the second field, and so on, until the array is exhausted or the record is full.

Syntax 2 uses a DynArray named *ar*, where each index is a field name or a field number, and the corresponding item is the field value.

This method fails if an attempt is made to copy an unassigned array element or if the structures do not match. If there are more elements in the array than fields in the record, the extra elements are ignored. Use **insertRecord** to insert a blank record before using **copyFromArray** to copy a new record into an empty table.

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCCREC;OPAL_METH_TCCTAR;OPAL_METH_TCIREC;OPAL_METH_TCIAFR;OPAL_METH_TCIBER;',0,"Defaultoverview",)} Related Topics

copyFromArray example

The following example assumes that CUSTNAME.DB has three fields: Last Name, A20; First name, A20; and Telephone, A12. This code associates a TCursor with the *CustName* table, creates an array with three elements, inserts a new record in the table and uses **copyFromArray** to copy data from the array to the new record:

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
  tc TCursor
  aa Array[3] AnyType
endVar
aa[1] = "Corel"
aa[2] = "Frank"
aa[3] = "555-1212"
if tc.open("CustName.db") then  ; open table
  tc.edit()                     ; copyFromArray works only in Edit mode
  tc.insertRecord()             ; insert new record
  tc.copyFromArray(aa)          ; copy from array to table
  tc.endEdit()
else
  msgStop("Stop", "Couldn't open CustName.db.")
endIf
endMethod
```

copyRecord method

Copies a record from one TCursor into another TCursor.

Syntax

```
copyRecord ( const sourceTC TCursor ) Logical
```

Description

copyRecord copies the record pointed to by one TCursor into the record pointed to by another TCursor. The following code copies a record from the *sourceTC* TCursor into the *destinationTC* TCursor:

```
destinationTC.copyRecord(sourceTC)
```

The TCursor specified by *sourceTC* does not have to be in Edit mode; however, the destination TCursor must be in Edit mode. This method fails if any field in the source record cannot be converted to the data type of the corresponding field in the destination record. This method returns True if it succeeds; otherwise, it returns False.

Note

- You cannot use **copyRecord** to copy a record into an empty table. To copy a new record into an empty table, use **insertRecord**.

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCCTAR;OPAL_METH_TCIAFR;OPAL_METH_TCIBER;OPAL_METH_TCIREC;',0,"Defaultoverview",)} Related Topics

copyRecord example

The following example uses a TCursor to scan the *Orders* table for sales posted within the last 10 days and copies them to the *NewOrdrs* table in the active directory. This code is attached to the **pushButton** method for the *getNewOrders* button:

```
; getNewOrders::pushButton
method pushButton(var eventInfo Event)
var
  ordTC,
   newOrdTC    TCursor
  tvNewOrds   TableView
endVar


ordTC.open("Orders.db")
newOrdTC.open("NewOrdrs.db")
newOrdTC.edit()               ; copyRecord only works in Edit mode.

; Scan Orders.db table for records posted in the last ten days.
scan ordTC for ordTC."Sale Date" >= (today() - 10) and
               ordTC."Sale Date" <= today() :
  newOrdTC.insertRecord()      ; Insert a new record in NewOrdrs.db.
  newOrdTC.copyRecord(ordTC)   ; Copy from Orders.db into NewOrdrs.db.
endScan
newOrdTC.endEdit()             ; End Edit mode for TCursor.


tvNewOrds.open("NewOrdrs.db")  ; Display the table.
endMethod
```

copyToArray method

Copies a record's fields to an array.

Syntax

```
1. copyToArray ( var ar Array[ ] AnyType ) Logical
2. copyToArray ( var ar DynArray[ ] AnyType ) Logical
```

Description

copyToArray copies a record's fields to the elements of an array specified by *ar*. You must declare the array as an AnyType type, or another type that matches each field in the table.

In Syntax 1, where *ar* is a fixed or resizeable array, the value of the first field is copied to the first element of the array, the value of the second field to the second element, and so on. If the array is resizeable, it expands to hold the number of fields in the record. If the array is fixed, it holds as many fields as possible, and discards the rest.

If Syntax 2, where *ar* is a dynamic array (DynArray), index values correspond to the field names and DynArray values correspond to field values.

ar [fieldName] = fieldValue

The record number field and any display-only or calculated fields that appear in a table's Form window are not copied to the array.

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCCFAR;',0,"Defaultoverview",)} Related Topics

copyToArray example

The following example assumes that a form has a table frame named CUSTOMER that is bound to CUSTOMER.DB. When the user attempts to delete a CUSTOMER record, this code uses **copyToArray** and **copyFromArray** to copy the record to an archive table (CUSTARC.DB). If CUSTARC.DB cannot be opened, this code informs the user and does not delete the record. The following code is attached to the built-in **action** method:

```
; CUSTOMER::action
method action(var eventInfo ActionEvent)
var
  tcOrig, tcArc TCursor
  arcRec Array[] AnyType
endVar


if eventInfo.id() = DataDeleteRecord then ; when user deletes a record
  if thisForm.Editing = True then         ; if form is in Edit mode
    disableDefault                        ; don't delete the record


                                          ; ask for confirmation
    if msgQuestion("Confirm", "Delete record?") = "Yes" then


      tcOrig.attach(CUSTOMER)             ; sync TCursor to UIObject
      tcOrig.copyToArray(arcRec)          ; store the record in arcRec
      if tcArc.open("CustArc.db") then    ; True if tcArc can open CustArc
        tcArc.edit()                      ; copyFromArray requires Edit
        tcArc.insertAfterRecord()         ; create a new record
        tcArc.copyFromArray(arcRec)       ; copy arcRec to new record
        enableDefault                     ; delete the record in Customer
      else                                ; can't open Customer TCursor
        msgStop("Stop!", "Sorry, Can't archive record.")
      endIf
    else                                  ; user didn't confirm dialog box
      message("Record not deleted.")
    endIf


  else                                    ; not in Edit mode
    msgStop("Stop!", "Press F9 to edit data.")
  endIf
endIf
endMethod
```

createIndex method

Creates an index for a table.

Syntax

```
1. createIndex ( const attrib DynArray[ ] AnyType, const fieldNames Array[ ] String ) Logical
2. createIndex ( const attrib DynArray[ ] AnyType, const fieldNums Array[ ] SmallInt ) Logical
```

Description

createIndex creates an index using attributes specified in a dynamic array (DynArray) named *attrib* and the field names (or numbers) specified in an Array named *fieldNames* (or *fieldNums*). This method is provided as an alternative to the **index** structure. It is especially useful when you don't know the index structure beforehand (e.g., when the information is supplied by the user).

Each key of the DynArray must be a string. You do not have to include all the keys to use **createIndex**. Any key you omit is assigned the corresponding default value.

The following table displays the key strings and their corresponding values:

String value	Description
MAINTAINED	If True, the index is incrementally maintained. That is, after a table is changed, only that portion of the index affected by the change is updated. If False, Corel Paradox does not maintain the index automatically. Maintained indexes typically result in better performance. Default = False (Corel Paradox tables only).
PRIMARY	If True, the index is a primary index. If False, it's a secondary index. Default = False (Corel Paradox tables only).
CASEINSENSITIVE	If True, the index ignores differences in case. If False, it considers case. Default = False (Corel Paradox tables only).
DESCENDING	If True, the index is sorted in descending order, from highest values to lowest. If False, it is sorted in ascending order. Default = False.
UNIQUE	If True, records with duplicate values in key fields are ignored. If False, duplicates are included and available.
IndexName	A name used to identify this index. No default value, unless you're creating a secondary, case-sensitive index on a single field, in which case the default value is the field name. For dBASE tables, the index name must be a valid DOS filename. If you do not specify an extension, .NDX is added automatically.
TagName	The name of the index tag associated with the index specified in *indexName* (dBASE tables only).

For more information on indexes, see About keys and indexes in tables in the User's Guide Help.

Examples

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCDROP;OPAL_METH_TBINDE;OPAL_METH_TBSIND;OPAL_METH_TBUSIND;',0,"Defaultoverview",)} **Related Topics**

createIndex method examples

createIndex example 1

The following example builds a maintained secondary index for a Corel Paradox table named CUSTOMER.DB. If the *Customer* table cannot be found or locked, this code aborts the operation:

```
method pushButton(var eventInfo Event)
var
   tbCust         Table
   stTbName         String
   tcCust         TCursor
   arFieldNames   Array[3] String
   dyAttrib         DynArray[]AnyType
endVar


stTbName        = "Customer.db"


arFieldNames[1] = "Customer No"
arFieldNames[2] = "Name"
arFieldNames[3] = "Street"


dyAttrib["PRIMARY"]    = False
dyAttrib["MAINTAINED"] = True
dyAttrib["IndexName"]  = "NumberNameStreet"


if isTable(stTbName) then
   tbCust.attach(stTbName)
   tbCust.setExclusive()


   if tcCust.open(tbCust) = FALSE then
      msgStop("Stop!", "Can't lock " + stTbName + " table.")
      return
   endif


   if not tcCust.createIndex(dyAttrib, arFieldNames) then
      errorShow()
   endif


; This createIndex statement has the same effect
; as the following INDEX structure:


 {


  INDEX "Customer.db"
    MAINTAINED
    ON "Customer No", "Name", "Street"
    TO "NumberNameStree"
  ENDINDEX


  }


else
  msgStop("Stop!", "Can't find " + stTbName + " table.")
endIf


endMethod
```

createIndex example 2

The following example builds a unique index named CITYSTAT.NDX for the dBASE table named CUSTOMER.DBF:

```
; cityStateIndex::pushButton
method pushButton(var eventInfo Event)
var
   tbCust      Table
   stTbName      String
   tcCust         TCursor
   arFieldNames   Array[1] String
   dyAttrib      DynArray[]AnyType
endVar

stTbName        = "Cust.dbf"

arFieldNames[1] = "CITY"

dyAttrib["UNIQUE"]     = True
dyAttrib["MAINTAINED"] = True

; A dBASE index name must be a valid DOS filename.
; If an extension is omitted, .NDX is appended automatically.

dyAttrib["IndexName"] = "City"



if isTable(stTbName) then
   tbCust.attach(stTbName)
   tbCust.setExclusive()
   if tcCust.open(tbCust) = False then
      msgStop("Stop!", "Can't lock " + stTbName + " table.")
      return
     endif

  tcCust.createIndex(dyAttrib, arFieldNames)
; This createIndex statement has the same effect
; as the following INDEX structure:
{
  INDEX "Cust.dbf"
       UNIQUE
    ON "CITY", "STATE_PROV"
    TO "CityStat"
  ENDINDEX
}

else
  msgStop("Stop!", "Can't find " + stTbName + " table.")

endif

endMethod
```

cSamStd method

Returns the sample standard deviation of a table's column.

Syntax

```
1. cSamStd ( const fieldName String ) Number
2. cSamStd ( const fieldNum SmallInt ) Number
```

Description

cSamStd returns the sample standard deviation for the column of numeric fields specified by *fieldName* or *fieldNum*. This method respects the limits of restricted views displayed in a linked table frame or multi-record object. **cSamStd** handles blank values as specified in the **blankAsZero** setting for the session.

The sample standard deviation calculation is based on the sample variance and uses the following formula:

sqrt(*TCursor*.cVar(*FieldName*) * (*n*/(*n*-1)))

where

(variance = *TCursor*.cVar(*fieldName*) and n = TCursor.cCount(fieldName))

Throughout the retry period, this method attempts to place a read lock on the table. If a lock cannot be placed, the method fails.

The population standard deviation is calculated using the **cStd** method.

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCCAVE;OPAL_METH_TCCCNT;OPAL_METH_TCCMAX;OPAL_METH_TCCMIN;OPAL_METH_TCCNPV;OPAL_METH_TCCSVA;OPAL_METH_TCCSTD;OPAL_METH_TCCSUM;OPAL_METH_TCCVAR;',0,"Defaultoverview",)} **Related Topics**

cSamStd example

The following example calculates the sample standard deviation of two fields in the *Answer* table. This code is attached to the **pushButton** method for *showSamStd*:

```
; showSamStd::pushButton
method pushButton(var eventInfo Event)
var
  empTC TCursor
  tblName String
  CalcSalary, CalcYears Number
endVar
tblName = "Answer"
if empTC.open(tblName) then
  CalcSalary = empTC.cSamStd("Salary")  ; get sample std deviation for salaries
  CalcYears  = empTC.cSamStd(2)         ; assume "Years in service" is field 2
  msgInfo("Sample Std Deviation",       ; display info in a dialog box
          "Salaries : " + String(CalcSalary) + "\n" +
          "Years in service : " + String(CalcYears))
else
  msgInfo("Sorry", "Can't open " + tblName + " table.")
endIf
endMethod
```

cSamVar method

Returns the sample variance in a column of fields.

Syntax

```
1. cSamVar ( const fieldName String ) Number
2. cSamVar ( const fieldNum SmallInt ) Number
```

Description

cSamVar returns the sample variance of the values in a column of fields. This method respects the limits of restricted views set by **setRange** or **setGenFilter**. This method handles blank values as specified in the **blankAsZero** setting for the session.

Throughout the retry period, this method attempts to place a read lock on the table. If a lock cannot be placed, the method fails.

The sample variance is calculated using this formula:

TCursor.cVar(*fieldName*) * (*n*/(*n*-1))

(n = TCursor.cCount(fieldName))

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCCAVE;OPAL_METH_TCCCNT;OPAL_METH_TCCMAX;OPAL_METH_TCCMIN;OPAL_METH_TCCNPV;OPAL_METH_TCCSST;OPAL_METH_TCCSTD;OPAL_METH_TCCSUM;OPAL_METH_TCCVAR;',0,"Defaultoverview",)} Related Topics

cSamVar example

The following example calculates the sample variance of two different fields in the *Answer* table. This code is attached to the **pushButton** method for *showSamVar*:

```
; showSamVar::pushButton
method pushButton(var eventInfo Event)
var
  empTC TCursor
  tblName String
  CalcSalary, CalcYears Number
endVar
tblName = "Answer"
if empTC.open(tblName) then
  CalcSalary = empTC.cSamVar("Salary") ; get sample variance for salaries
  CalcYears  = empTC.cSamVar(2)        ; assume "Years in service" is field 2
  msgInfo("Sample Variance",           ; display info in a dialog box
          "Salaries : " + String(CalcSalary) + "\n" +
          "Years in service : " + String(CalcYears))
else
  msgInfo("Sorry", "Can't open " + tblName + " table.")
endIf
endMethod
```

cStd method

Returns the standard deviation of the values in a column.

Syntax

```
1. cStd ( const fieldName String ) Number

2. cStd ( const fieldNum SmallInt ) Number
```

Description

cStd returns the population standard deviation of the values in a column of fields. The calculation is based on the variance. This method respects the limits of restricted views set by **setRange** or **setGenFilter**. This method handles blank values as specified in the **blankAsZero** setting for the session.

Throughout the retry period, this method attempts to place a read lock on the table. If a lock cannot be placed, the method fails.

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCCAVE;OPAL_METH_TCCCNT;OPAL_METH_TCCMAX;OPAL_METH_TCCMIN;OPAL_METH_TCCNPV;OPAL_METH_TCCSST;OPAL_METH_TCCSVA;OPAL_METH_TCCSUM;OPAL_METH_TCCVAR;',0,"Defaultoverview",)} Related Topics

cStd example

In the following example, the **pushButton** method for *thisButton* calculates the population standard deviation for two separate fields. The results are displayed in a dialog box:

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
  tc TCursor
  test1, test2 Number
endVar
tc.open("scores.dbf")
test1 = tc.cStd("Test1")
test2 = tc.cStd(2)             ; assumes Test2 is field 2


; show results in a dialog
msgInfo("Standard Deviation",
        "Test1 results : " + String(test1) + "\n" +
        "Test2 results : " + String(test2))


endMethod
```

cSum method

Returns the sum of the values in a column of fields.

Syntax

```
1. cSum ( const fieldName String ) Number
2. cSum ( const fieldNum SmallInt ) Number
```

Description

cSum returns the sum of the values in a column of fields. This method respects the limits of restricted views set by **setRange** or **setGenFilter**. This method handles blank values as specified in the **blankAsZero** setting for the session.

Throughout the retry period, this method attempts to place a read lock on the table. If a lock cannot be placed, the method fails.

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCCAVE;OPAL_METH_TCCCNT;OPAL_METH_TCCMAX;OPAL_METH_TCCMIN;OPAL_METH_TCCNPV;OPAL_METH_TCCSST;OPAL_METH_TCCSVA;OPAL_METH_TCCSTD;OPAL_METH_TCCVAR;',0,"Defaultoverview",)} Related Topics

cSum example

In the following example, the **pushButton** method for *sumOrders* calculates totals for two fields in ORDERS.DB:

```
; sumOrders::pushButton
method pushButton(var eventInfo Event)
var
  orderTC TCursor
  orderTotal, amtPaid Number
  tblName String
endVar
tblName = "Orders"
if orderTC.open(tblName) then
  orderTotal = orderTC.cSum("Total Invoice")  ; get sum for Total Invoice field
  amtPaid    = orderTC.cSum(7)                ; assumes Amount Paid is field 7
  msgInfo("Order Totals",
          "Total Orders : " + String(orderTotal) + "\n" +
          "Total Receipts : " + String(amtPaid))
else
  msgInfo("Sorry", "Can't open " + tblName + " table.")
endIf
endMethod
```

currRecord method

Reads the active record into the record buffer.

Syntax

currRecord ( ) Logical

Description

currRecord reads the values in the active record of the underlying table into the record buffer. **currRecord** cancels any unposted changes to the TCursor. This method ensures that you're using the most recent version of the record on a network.

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCHOME;OPAL_METH_TCEND;OPAL_METH_TCNERE;OPAL_METH_TCPRRE;OPAL_METH_TCSKIP;OPAL_METH_TCMTRE;',0,"Defaultoverview",)} Related Topics

currRecord example

The following example is part of a system that processes concert ticket orders. This code determines which artist the customer wants to see and how many seats the customer needs.

```
; updateSeats::pushButton
method pushButton(var eventInfo Event)
   var
      tcConcert      TCursor
      siSeatsNeeded,
      siCustSeats      SmallInt
      stArtist      String
   endVar


   ; Call a custom method to find out which artist
   ; the customer wants to see.
   stArtist = getArtistName()
   tcConcert.open("concerts")
   tcConcert.locate("Artist", stArtist)


   if tcConcert.SoldOut = True then
         msgStop("Sorry", "Sold out")
         return
   else


      ; Call a custom method to find out how many seats
      ; the customer needs (this may take awhile).
      siCustSeats = getCustSeats()


      ; Meanwhile, other customers may have ordered seats for this
      ; concert, so read current values into the record buffer.
      tcConcert.currRecord()


      if tcConcert.Seats >= siCustSeats then
            processOrder() ; Call a custom method to process the order.
      else
            notEnoughSeats() ; Call a custom method.
      endIf
   endIf
endMethod
```

cVar method

Returns the variance of the values in a column of numeric fields.

Syntax

```
1. cVar ( const fieldName String ) Number
2. cVar ( const fieldNum SmallInt ) Number
```

Description

cVar returns the population variance of values in a column of numeric fields. This method respects the limits of restricted views set by **setRange** or **setGenFilter**. **cVar** handles blank values as specified in the **blankAsZero** setting for the session.

Throughout the retry period, this method attempts to place a read lock on the table. If a lock cannot be placed, the method fails.

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCCAVE;OPAL_METH_TCCCNT;OPAL_METH_TCCMAX;OPAL_METH_TCCMIN;OPAL_METH_TCCNPV;OPAL_METH_TCCSVA;OPAL_METH_TCCSST;OPAL_METH_TCCSTD;OPAL_METH_TCCSUM;',0,"Defaultoverview",)} **Related Topics**

cVar example

In the following example, the **pushButton** method for *thisButton* calculates the population variance deviation for two fields. The results are displayed in a dialog box:

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
  myTable TCursor
  test1, test2 Number
endVar
myTable.open("scores.dbf")
test1 = myTable.cVar("Test1")     ; get Test1 cVar
test2 = myTable.cVar(2)           ; assumes Test2 is field 2
msgInfo("Population Variance",
        "Test1 results : " + String(test1) + "\n" +
        "Test2 results : " + String(test2))
endMethod
```

deleteRecord method

Deletes the record pointed to by a TCursor.

Syntax

```
deleteRecord ( ) Logical
```

Description

deleteRecord deletes the record pointed to by a TCursor without asking for confirmation. Deleted Corel Paradox records cannot be retrieve, but deleted dBASE records can. The table must be in Edit mode.

If the specified record is contained in a dBASE table and is locked or has already been deleted by another user, this method fails.

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCCANC;OPAL_METH_TCEMPT;OPAL_METH_TCUNDELETERECORD;',0,"Defaultoverview",)} Related Topics

deleteRecord example

In the following example, the **pushButton** method for the *checkIOU* button determines whether a debt has been paid. If the record has been marked as paid, this code uses **deleteRecord** to delete the record:

```
; checkIOU::pushButton
method pushButton(var eventInfo Event)
var
  iou TCursor
  searchName String
endVar
searchName = "Hall"
iou.open("iou.db")
iou.edit()
if iou.locate("Name", searchName) then
  if iou."paid" = "Yes" then
    iou.deleteRecord()           ; delete the active record
    message(searchName + " deleted")
  else
    sendBill()                   ; run custom procedure
  endIf
else
   msgStop("Stop", "Couldn't find " + searchName)
endIf
endMethod
```

didFlyAway method

Reports whether a key value change moved the active record moved to a different position in the table.

Syntax

```
didFlyAway ( ) Logical
```

Description

didFlyAway returns True if the most recent call to **unlockRecord** caused the record to move to a different position in the table; otherwise, it returns False. This method is only accurate if the **setFlyAwayControl** method has been set to True; otherwise, **didFlyAway** returns always False.

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCSETFLYAWAYCONTROL;OPAL_METH_TCUNREC;',0,"Defaultoverview",)} Related Topics

didFlyAway example

The following example demonstrates how **setFlyAwayControl** affects the position of a TCursor after a call to **unlockRecord**, and under what circumstances **didFlyAway** returns True:

```
; demoButton::pushButton
method pushButton(var eventInfo Event)
var
  tc TCursor
endvar


tc.open("MyTable.db")


; Assume that MyTable.db has the following
; values in its only key field, "Customer No" :
;  Record#  Customer No
;     1        110
;     2        120   ; the code below changes this value to 145
;     3        130
;     4        140
                     ; which moves the record to this position
;     5        150


tc.setFlyAwayControl(Yes) ; Enable flyaway tracking.


if tc.locate("Customer No", 120) then
  tc.edit()


  ; Change the key value so that the record
  ; changes relative position.
  tc."Customer No" = 145


  tc.unlockRecord()       ; Unlock the record.


  ; The dialog box displays True because the new key value
  ; changes the record's relative position in the table.
  msgInfo("Did 145 fly away?", tc.didFlyAway())

else
  message("120 not found.")
endIf


endMethod
```

dmAttach method

Associates a TCursor with a table in the data model.

Syntax

```
dmAttach ( const dmTableName String ) Logical
```

Description

dmAttach associates a TCursor with the table specified by *dmTableName*. The table must be in the data model. This method returns True if successful; otherwise, it returns False.

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCATT;OPAL_METH_TCOPEN;OPAL_METH_FODMATTACH;',0,"Defaultoverview",)} **Related Topics**

dmAttach example

The following example uses **dmAttach** to open a TCursor to a table in the data model. The TCursor respects the restricted view of the data model. The code uses **cSum** to gather information stored in the string variables *s1, s2,* and *s3.* The information is displayed in a dialog box.

```
;btnCustomerSummary :: pushButton
method pushButton(var eventInfo Event)
var
      tc   TCursor
      s1   String
      s2   String
      s3   String
endVar
tc.dmAttach("Orders.db")
s1 = string(tc.cSum("Total Invoice"))
s2 = string(tc.cSum("Amount Paid"))
s3 = string(tc.cSum("Balance Due"))


msgInfo("Customer Summary",
"Total Orders = " + s1 +
"\nTotal Paid = " + s2 +
"\nTotal Due = " + s3)
endMethod
```

dropGenFilter method

Removes the filter criteria associated with a TCursor.

Syntax

dropGenFilter ( ) Logical

Description

dropGenFilter removes the filter criteria associated with a TCursor. Any indexes and tags remain in effect in the unfiltered TCursor.

Examples

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCGETGENFILTER;OPAL_METH_TCSETGENFILTER;OPAL_METH_TCSETRANGE;OPAL_METH_UIDROPGENFILTER;',0,"Defaultoverview",)} Related Topics

dropGenFilter method examples

Example1 Using a TCursor to calculate averages in a table

Example2 Using the **pushButton** method to attach a TCursor

dropGenFilter example 1

The following example attaches a TCursor to a table frame that is bound to the *Orders* table. This code calculates the average total invoice amount for the entire table by calling **dropGenFilter** to remove any user-defined or automatically generated filter criteria. The call to **dropGenFilter** operates on the TCursor only—it does not affect the table frame.

```
; btnCalAvgInvoice::pushButton
method pushButton(var eventInfo Event)
var
      ordersTC   TCursor
      nuAvgInvoice   Number
endVar
   ordersTC.attach(Orders)    ; Attach to the Orders table frame.
   ordersTC.dropGenFilter()   ; Remove any filters on the TCursor.

nuAvgInvoice = ordersTC.cAverage("Total Invoice")
nuAvgInvoice.view("Average Total Invoice:")
endMethod
```

dropGenFilter example 2

In the following example, a form contains a button named *btnCascadeDelete*. The **pushButton** method for *btnCascadeDelete* attaches a TCursor to a child table (the UIObject LINEITEM), uses **dropGenFilter** to ensure the TCursor can see all the child records, moves the TCursor to the first record, and puts the TCursor in edit mode. A **while** loop deletes all the child records and then the form is placed in edit mode and the parent record is deleted.

```
;btnCascadeDelete::pushButton
method pushButton(var eventInfo Event)
var
      tc               TCursor
      siCounter   SmallInt
endVar


tc.attach(LINEITEM)  ;Attach to detail table.
tc.dropGenFilter()   ;Drop any user set filters.
tc.home()            ;Make sure TCursor is on first record.


tc.edit()
while not tc.eot()        ;If there are any child
   tc.deleteRecord()      ;records, delete all of them.
endWhile


edit()                    ;Make sure form is in edit mode.
Order_No.deleteRecord()   ;Delete the parent record.
endMethod
```

dropIndex method

Deletes a specified index file or tag.

Syntax

```
1. (Corel Paradox tables) dropIndex ( const indexName String ) Logical
2. (dBASE tables) dropIndex ( const indexName String [ , const tagName String ] ) Logical
```

Description

dropIndex deletes a specified index file or tag. You can't delete an index that's in use.

In a Corel Paradox table, *indexName* is required to specify a secondary index. You can't use a TCursor to drop the primary index of a Corel Paradox table.

In a dBASE table, you can use *indexName* to specify an .NDX file, or use *indexName* and *tagName* to specify an .MDX file and an index tag.

Note

- You must open the TCursor on a Table variable that has called the Table method **setExclusive** (*before* opening the table) before calling **dropIndex**.

For more information about indexes, see About keys and indexes in tables in the User's Guide help.

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCSWIT;',0,"Defaultoverview",)} Related Topics

dropIndex example

In the following example, the **pushButton** method for a button deletes the *CustName* tag from an .MDX file:

```
method pushButton(var eventInfo Event)
var
   tc1  TCursor
   tb1  Table
endVar


if isTable("Sales.dbf") then
tb1.attach("Sales.dbf") ; Sales.dbf is a dBASE table
tb1.setExclusive (Yes)
tc1.open(tb1)


      ; delete CustName tag from index2 file
   if tc1.dropIndex("index2.mdx", "CustName") then
      msgInfo("", "custname dropped")
   else
      errorShow("Could not drop index.")
   endIf
else
  msgStop("Stop!", "Could not find Sales.dbf table.")
endIf


endMethod
```

edit method

Places a TCursor in Edit mode.

Syntax

```
edit ( ) Logical
```

Description

edit places a TCursor in Edit mode allowing you to modify the active record. To remain in Edit mode, move off the record or use **postRecord** or **unlockRecord** to accept changes. To leave Edit mode, use **cancelEdit** to cancel changes to the record or use **endEdit** to accept changes.

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCCANC;OPAL_METH_TCENDED;OPAL_METH_TCPOSTRECORD;',0,"Defaultoverview",)} Related Topics

edit example

The following example creates an array and uses **copyFromArray** to copy its contents to a new record in the *CustName* table. Because the TCursor must be in Edit mode before the new record is inserted, this code uses **edit** to begin editing the table. After the new record is inserted, **endEdit** accepts changes and exits Edit mode:

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
  tc TCursor
  aa Array[3] AnyType
endVar
aa[1] = "Corel"
aa[2] = "Frank"
aa[3] = "555-1212"
if tc.open("custname.db") then  ; open table
  tc.edit()                     ; put TCursor in Edit mode
  tc.insertRecord()             ; insert new record
  tc.copyFromArray(aa)          ; copy from array to table
  tc.endEdit()                  ; end Edit mode
else
  msgStop("Stop", "Couldn't open Custname.db.")
endIf
endMethod
```

empty method

Deletes all records from a table.

Syntax

empty ( ) Logical

Description

empty deletes all records from a table without asking for confirmation. If the TCursor is associated with a dBASE table, the records are flagged as deleted and the table is compacted (if possible). The TCursor does not have to be in Edit mode to empty records, but a write lock is required. This operation cannot be undone.

empty removes information from the table, but does not delete the table itself. Compare this method to **delete**, which does delete the table.

empty first tries to gain exclusive rights to the table. If it can't, it tries to place a write lock on the table.

If **empty** gains exclusive rights, it deletes all records in the table at once. If a write lock is placed on the table, **empty** must delete each record individually.

If **empty** gains exclusive rights to a dBASE table, all records are deleted and the table is compacted. If a write lock is placed on the table, this method flags all records as deleted, but does not remove them from the table. (Records can be undeleted from a dBASE table if they have not been removed with the **compact** method.)

 **Example**

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCDREC;OPAL_METH_TCNREC;OPAL_METH_TBDELETE;OPAL_METH_TBEMPT;',0,"Defaultoverview",)} Related Topics

empty example

The following example prompts the user for confirmation before deleting all records from the *Scratch* table. If the user does not confirm the action, this code uses **nRecords** to determine how many records exist in SCRATCH.DB:

```
; tblEmpty::pushButton
method pushButton(var eventInfo Event)
var
  tblName String
  tc TCursor
endVar


tblName = "Scratch.db"
if isTable(tblName) then
  tc.open(tblName)
  if msgQuestion("Confirm", "Empty " + tblName + " table?") = "Yes" then
    tc.empty()
    message("All " + tblName + " records have been deleted.")
  else
    message(tblname + " has " + String(tc.nRecords()) + " records.")
  endIf
else
  msgInfo("Error", "Can't find " + tblName + " table.")
endIf
endMethod
```

end method

Moves a TCursor to the table's last record.

Syntax

end ( ) Logical

Description

end sets the active record (and the record buffer) to the table's last record.

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCHOME;OPAL_METH_TCNERE;OPAL_METH_TCPRRE;OPAL_METH_TCCURE;OPAL_METH_TCSKIP;OPAL_METH_TCMTRE;',0,"Defaultoverview",)} Related Topics

end example

The following example uses **end** to move a TCursor to the last record in the *Orders* table. The information in the last record is displayed in a dialog box.

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
  tc TCursor
endVar
tc.open("Orders.db")        ; open tc for Orders table
tc.end()                    ; move to the last record in the table
                            ; display info in last record
msgInfo("Customer No " + tc."Customer No",
        "Outstanding balance: " + tc."Balance Due")


endMethod
```

endEdit method

Exits Edit mode and accepts changes to the active record.

Syntax

```
endEdit ( ) Logical
```

Description

endEdit exits Edit mode and accepts changes to the active record. This method does not close the TCursor. Changes to previous records are committed or canceled as the user navigates the table.

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCCANC;OPAL_METH_TCCOED;',0,"Defaultoverview",)}

Related Topics

endEdit example

The following example creates an array and uses **copyFromArray** to copy its contents to a new record in the *CustName* table. Because *CustName* must be in Edit mode before the new record is inserted, this code uses **edit** to begin editing the table. When the new record is inserted, this code uses **endEdit** to exit Edit mode:

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
  tc TCursor
  aa Array[3] AnyType
endVar
aa[1] = "Corel"
aa[2] = "Frank"
aa[3] = "555-1212"
if tc.open("custname.db") then  ; open table
  tc.edit()                     ; put TCursor in Edit mode
  tc.insertRecord()             ; insert new record
  tc.copyFromArray(aa)          ; copy from array to table
  tc.endEdit()                  ; end Edit mode
else
  msgStop("Stop", "Couldn't open Custname.db.")
endIf
endMethod
```

enumFieldNames method

Fills an array with the table's field names.

Syntax

```
enumFieldNames ( const fieldArray Array[ ] String ) Logical
```

Description

enumFieldNames fills an array named *fieldArray* with a table's field names. If *fieldArray* is resizeable, it expands to hold the field names. If *fieldArray* is fixed, it holds as many field names as possible and discards the rest. If *fieldArray* already exists, **enumFieldNames** overwrites it without asking for confirmation.

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCENUMFIELDNAMESININDEX;',0,"Defaultoverview",)}

Related Topics

enumFieldNames example

In the following example, the **pushButton** method for the *btnEnumFields* button stores field names in a resizeable array and uses **view** to display the contents of the array:

```
; enumFields::pushButton
method pushButton(var eventInfo Event)
var
  tc TCursor
  fieldNames Array[] String     ; field names for tables are always strings
endVar
if tc.open("orders.db") then
  tc.enumFieldNames(fieldNames) ; load fieldNames with names of Orders.db fields
  fieldNames.view()             ; display field names in a dialog box
else
  msgStop("Stop", "Couldn't open Orders.db.")
endIf

endMethod
```

enumFieldNamesInIndex method

Fills an array with a table index's field names.

Syntax

1. (Corel Paradox tables) **enumFieldNamesInIndex (** [const ***indexName*** String,] var ***fieldArray*** Array[ ] String **)** Logical

2. (dBASE tables) **enumFieldNamesInIndex (** [const ***indexName*** String [, const ***tagName*** String,] var ***fieldArray*** Array[ ] String **)** Logical

Description

enumFieldNamesInIndex fills *fieldArray* with the names of the fields in a table's index, as specified in *indexName*. If *indexName* is omitted, this method uses the current index. If *fieldArray* is resizeable, it expands to hold all of the field names. If *fieldArray* is fixed, it holds as many field names as possible, and discards the rest. If *fieldArray* already exists, this method overwrites it without asking for confirmation.

In a dBASE table, you can use the optional argument *tagName* to specify an index tag within an .MDX file.

For more information on indexes, see About keys and indexes in tables in the User's Guide help.

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCENUMINDEXSTRUCT;',0,"Defaultoverview",)} Related Topics

enumFieldNamesInIndex example

In the following example, the **pushButton** method for the *enumIndex* button stores field names in a resizeable array and uses **view** to display the contents of the array:

```
; enumIndex::pushButton
method pushButton(var eventInfo Event)
var
  tc TCursor
  fieldNames Array[] String
endVar
if tc.open("Sales.dbf") then
  ; load fieldNames array with field names in the byDate index
  tc.enumFieldNamesInIndex("DateIndx.MDX", "byDate", fieldNames)
  ; display the index field names for byDate in DateIndx
  fieldNames.view()
else
  msgStop("Stop", "Couldn't open Sales.dbf.")
endIf
endMethod
```

enumFieldStruct method

Lists the field structure of a TCursor.

Syntax

```
1. enumFieldStruct ( const tableName String ) Logical
2. enumFieldStruct ( inMem TCursor ) Logical
```

Description

enumFieldStruct lists the field structure of a TCursor. Syntax 1 creates a Corel Paradox table; Syntax 2 stores the information in a TCursor variable.

Syntax 1 creates a Corel Paradox table *tableName*. If *tableName* already exists, this method overwrites it without asking for confirmation. If *tableName* is open, this method fails. You can also include an alias or path in *tableName*. If an alias or path is not specified, Corel Paradox creates the table in the working directory. You can supply *tableName* to the **struct** option in a **create** statement to borrow that table's field structure (including primary keys and validity checks) for use in the new table.

In Syntax 2, the structure information is stored in the TCursor variable *inMem* that you pass as an argument. Syntax 2 results in faster performance because the information is stored in system memory.

The following table displays the structure of the table in Syntax 1 or the TCursor in Syntax 2:

Field	Type	Description
Field Name	A31	Specifies the name of field
Type	A31	Specifies the data type of field
Size	S	Specifies the size of field
Dec	S	Specifies the number of decimal places, or 0 if field type doesn't support decimal places
Key	A1	Specifies whether the field is a key (* = key field, blank = not key field)
_Required Value	A1	Specifies whether the field is required (T = required, N (or blank) = Not required)
_Min Value	A255	Specifies the field's minimum value
_Max Value	A255	Specifies the field's maximum value
_Default Value	A255	Specifies the field's default value
_Picture Value	A175	Specifies the field's picture
_Table Lookup	A255	Specifies the name of lookup table (including the full path if the lookup table is not in :WORK:)
_Table Lookup Type	A1	Specifies the type of lookup table 0 (or blank) = no lookup table, 1 = Current field + private 2 = All corresponding + no help 3 = Just current field + help and field 4 = All corresponding + help
_Invariant Field ID	S	Specifies the field's ordinal position in table (first field = 1, second field = 2, etc.)

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCENUMFIELDNAMES;OPAL_METH_TCENUMFIELDNAMESININDEX;OPAL_METH_TCENUMINDEXSTRUCT;OPAL_METH_TCENUMREFINTSTRUCT;OPAL_METH_TCENUMSECSTRUCT;',0,"Defaultoverview",)} Related Topics

enumFieldStruct example

The following example assumes that you want a new table named *NewCust* that is similar to the *Customer* table. It also assumes that you want all of the fields in *NewCust* to be required fields. The following code uses **enumFieldStruct** to load a new table (CUSTFLDS.DB) with the field-level information from *Customer*. The code then scans *CustFlds* and modifies the field definitions so that each record describes a required field. *CustFlds* is then supplied in the **struct** clause of a **create** statement.

```
; makeNewCust::pushButton
method pushButton(var eventInfo Event)
var
  newCustTbl Table
  tc   TCursor
  structName, sourceName String
endVar


structName = "CustFlds.db"
sourceName = "Customer.db"


if tc.open(sourceName) then


  tc.enumFieldStruct(structName)


  ; Point the TCursor to the CustFlds table.
  tc.open(structName)
  tc.edit()

  ; This loop scans through the CustFlds table and
  ; changes ValCheck definitions for every field.
  scan tc :
    tc."_Required Value" = 1    ; Make all fields required.
  endScan


  ; Now create NEWCUST.DB and borrow field names,
  ; ValChecks and key fields from CUSTFLDS.DB.
  newCustTbl = CREATE "NewCust.db"
                 STRUCT structName
               ENDCREATE


  ; NEWCUST.DB requires that all fields be filled.


else
  msgStop("Error", "Can’t get field structure for Customer table.")
endIf


endMethod
```

enumIndexStruct method

Lists the structure of a TCursor's secondary indexes.

Syntax

```
1. enumIndexStruct ( const tableName String ) Logical
2. enumIndexStruct ( inMem TCursor ) Logical
```

Description

enumIndexStruct lists the structure of a TCursor's secondary indexes. Syntax 1 creates a Corel Paradox table; Syntax 2 stores the information in a TCursor variable.

Syntax 1 creates the Corel Paradox table specified in *tableName*. For dBASE tables, this method lists the structure of the indexes associated with the table by the **usesIndexes** method. If *tableName* already exists, this method overwrites it without asking for confirmation. You can also include an alias or path in *tableName*. If an alias or path is not specified, Corel Paradox creates the table in the working directory. You can supply *tableName* to the **indexStruct** option in a **create** statement to borrow that table's field structure (including primary keys and validity checks) for use in the new table.

In Syntax 2, the structure information is stored in the TCursor variable *inMem* that you pass as an argument. Syntax 2 results in faster performance because the information is stored in system memory.

The following table displays the structure of the table in Syntax 1 or the TCursor in Syntax 2:

Field	Type	Description
infoHeader	A1	Specifies whether this record is a header for (and the data it contains is shared by) subsequent consecutive records that have a value of N in this field
szName	A255	Specifies the index name, including path
szTagName	A31	Specifies the tag name, no path (dBASE only)
szFormat	A31	Specifies the optional index type, e.g., BTREE, HASH
bPrimary	A1	Specifies whether the index is primary
bUnique	A1	Specifies whether the index is unique
bDescending	A1	Specifies whether the index is descending
bMaintained	A1	Specifies whether the index is maintained
bCaseInsensitive	A1	Specifies whether the index is case-sensitive
bSubset	A1	Specifies whether the index is a subset index (dBASE only)
bExpIdx	A1	Specifies whether the index is an expression index (dBASE only)
iKeyExpType	N	Specifies the key type of index expression (dBASE only)
szKeyExp	A220	Specifies the key expression for expression index (dBASE only)
szKeyCond	A220	Specifies the subset condition for subset index (dBASE only)
FieldNo	N	Specifies the ordinal position of key field in table
FieldName	A31	Specifies the name of key field

For more information on indexes, see About keys and indexes in tables in the User's Guide help.

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCENUMFIELDNAMES;OPAL_METH_TCENUMFIELDNAMESININDEX;OPAL_METH_TCENUMFIELDSTRUCT;OPAL_METH_TCENUMREFINTSTRUCT;OPAL_METH_TCENUMSECSTRUCT;',0,"Defaultoverview",)} **Related Topics**

enumIndexStruct example

The following example assumes that you want a new table named *NewCust* that is similar to the *Customer* table. It also assumes that you don't want to borrow referential integrity or security information. The following code uses **enumFieldStruct** and **enumIndexStruct** to generate two tables (CUSTFLDS.DB and CUSTINDX.DB). *CustFlds* and *CustIndx* are then supplied to the **struct** and **indexStruct** clauses of a **create** statement.

```
; makeNewCust::pushButton
method pushButton(var eventInfo Event)
var
  newcustTC Table
  custTC    TCursor
endVar

if custTC.open("Customer.db") then

  ; write field level information to CUSTFLDS.DB
  custTC.enumFieldStruct("CustFlds.db")

  ; write secondary index information to CUSTINDX.DB
  custTC.enumIndexStruct("CustIndx.db")

  ; now create NEWCUST.DB
  ; borrow field names, ValChecks, and key fields from CUSTFLDS.DB
  ; borrow secondary indexes from CUSTINDX.DB
  newcustTC = CREATE "NewCust.db"
                STRUCT "CustFlds.db"
                INDEXSTRUCT "CustIndx.db"
              ENDCREATE

else
  msgStop("Error", "Can't find Customer table.")
endIf

endMethod
```

enumLocks method

Creates a Corel Paradox table listing the locks currently applied to a table.

Syntax

```
enumLocks ( const tableName String ) LongInt
```

Description

enumLocks creates a Corel Paradox table specified by *tableName* that lists the number of locks on the specified table. If *tableName* already exists, this method overwrites it without asking for confirmation. If *tableName* is open, this method fails. For dBASE tables, this method lists only the lock you've placed (not all locks currently on the table). You can also include an alias or path in *tableName*; if an alias or path is not specified, Corel Paradox creates *tableName* in the working directory.

The following table describes the structure of *tableName* :

Field name	Field type	Description
UserName	A15	Specifies the user name
LockType	A32	Describes the type of lock (e.g., Table Write Lock)
NetSession	N	Specifies the net level session number
Session	N	Specifies the BDE session number (if the lock was placed by BDE)
RecordNumber	N	Specifies the record number (if the lock is a record lock; otherwise 0)

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCLOCK;OPAL_METH_TCLSTA;',0,"Defaultoverview",)}

Related Topics

enumLocks example

In the following example, the built-in **pushButton** method for the *showOrdersLcks* button creates a table listing the locks currently applied to ORDERS.DB:

```
; showOrdersLcks::pushButton
method pushButton(var eventInfo Event)
var
  tc TCursor
  tv TableView
endVar
if tc.open("Orders.db") then
  tc.enumLocks("OrderLck.db")  ; store Orders.db locks in OrderLck.db
  tv.open("OrderLck.db")       ; open OrderLck.db
else
  msgStop("Stop!", "Can't open Orders.db table")
endIf

endMethod
```

enumRefIntStruct method

Lists referential integrity information for a TCursor.

Syntax

```
1. enumRefIntStruct ( const tableName String ) Logical
2. enumRefIntStruct ( inMem TCursor ) Logical
```

Description

enumRefIntStruct lists referential integrity information for a TCursor. Syntax 1 creates a Corel Paradox table; Syntax 2 stores the information in a TCursor variable.

Syntax 1 creates the Corel Paradox table specified in *tableName*. If *tableName* is open, this method fails. If *tableName* already exists, this method overwrites it without asking for confirmation. You can also include an alias or path in *tableName*. If an alias or path is not specified, Corel Paradox creates the table in the working directory. You can supply *tableName* to the **refIntStruct** option in a **create** statement to borrow that table's field structure (including primary keys and validity checks) for use in the new table.

In Syntax 2, the structure information is stored in the TCursor variable *inMem* that you pass as an argument. Syntax 2 results in faster performance because the information is stored in system memory.

The following table displays the structure of the table in Syntax 1 or the TCursor in Syntax 2:

Field name	Type	Description
infoHeader	A1	Specifies whether the record is a header for (and the data it contains is shared by) subsequent consecutive records that have a value of N in this field
RefName	A31	Specifies the name to identify this referential integrity constraint
OtherTable	A255	Specifies the name (including path) of the other table in the referential integrity relationship
Slave	A1	Specifies whether the table is slave, not master (i.e., the table is dependent)
Modify	A1	Specifies the update rule (Y = Cascade, blank = Prohibit)
Delete	A1	Specifies the delete rule (blank = Prohibit). Corel Paradox does not support cascading deletes for Corel Paradox or dBASE tables.
FieldNo	N	Specifies the ordinal position of the field in this table involved in a referential integrity relationship
aiThisTabField	A31	Specifies the name of the field in this table involved in a referential integrity relationship
Other FieldNo	N	Specifies the ordinal position of the field in the other table involved in a referential integrity relationship
aiOthTabField	A31	Specifies the name of the field in the other table involved in a referential integrity relationship

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCENUMFIELDNAMES;OPAL_METH_TCENUMFIELDNAMESININDEX;OPAL_METH_TCENUMFIELDSTRUCT;OPAL_METH_TCENUMINDEXSTRUCT;OPAL_METH_TCENUMSECSTRUCT;OPAL_METH_FOENTAB;',0,"Defaultoverview",)} **Related Topics**

enumRefIntStruct example

The following example uses **enumRefIntStruct** to write CUSTOMER.DB referential integrity information to the *CustRef* table. The code supplies *CustRef* to the **refIntStruct** clause in a **create** statement:

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
  tc  TCursor
  tbl Table
endVar


tc.open("Customer.db")


; Write referential integrity information to CustRef.
tc.enumRefIntStruct("CustRef.db")
; Write field level information to CustFlds.
tc.enumFieldStruct("CustFlds.db")
tc.close()
; Now create NEWCUST.DB.
; Borrow field level information from CUSTFLDS.DB.
; Borrow referential integrity information from CUSTREF.DB.
tbl = CREATE "NewCust.db"
        STRUCT "CustFlds.db"
        REFINTSTRUCT "CustRef.db"
      ENDCREATE


endMethod
```

enumSecStruct method

Lists a TCursor's security information.

Syntax

```
1. enumSecStruct ( const tableName String ) Logical
2. enumSecStruct ( inMem TCursor ) Logical
```

Description

enumSecStruct lists the security information (access rights) of a TCursor. Syntax 1 creates a Corel Paradox table; Syntax 2 stores the information in a TCursor variable.

Syntax 1 creates the Corel Paradox table specified in *tableName*. For dBASE tables, this method lists the structure of the indexes associated with the table by the **usesIndexes** method. If *tableName* is open, this method fails. If *tableName* already exists, this method overwrites it without asking for confirmation. You can also include an alias or path in *tableName*. If an alias or path is not specified, Corel Paradox creates the table in the working directory. You can supply *tableName* to the **secStruct** option in a **create** statement to borrow that table's field structure (including primary keys and validity checks) for use in the new table.

In Syntax 2, the structure information is stored in the TCursor variable *inMem* that you pass as an argument. Syntax 2 results in faster performance because the information is stored in system memory.

The following table displays the structure of the table in Syntax 1 or the TCursor in Syntax 2:

Field name	Type	Description
infoHeader	A1	Specifies whether the record is a header for (and the data it contains is shared by) subsequent consecutive records that have a value of N in this field
iSecNum	N	Specifies the number to identify security description (first description = 1)
eprvTable	N	Specifies the table privilege value
eprvTableSym	A10	Specifies the table privilege name
iFamRights	N	Specifies the family rights value
iFamRightsSym	A10	Specifies the family rights name
szPassword	A31	Specifies the password
fldNum	N	Specifies the ordinal position of field in table
aprvFld	N	Specifies the field privilege value
aprvFldSym	A10	Specifies the field privilege name

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCENUMFIELDNAMES;OPAL_METH_TCENUMFIELDNAMESININDEX;OPAL_METH_TCENUMFIELDSTRUCT;OPAL_METH_TCENUMINDEXSTRUCT;OPAL_METH_TCENUMREFINTSTRUCT;',0,"Defaultoverview",)} Related Topics

enumSecStruct example

The following example creates a new table based on the security information that is associated with the *Secrets* table. This code uses **enumSecStruct** to write security information to the *SecInfo* table which is then used to create the *MySecrts* table:

```
method pushButton(var eventInfo Event)
var
  tc  TCursor
  tbl Table
endVar


; Associate tc with SECRETS.DB.
tc.open("Secrets.db")
; Write security information to SECINFO.DB.
tc.enumSecStruct("SecInfo.db")


; Now create MYSECRTS.DB.
; Borrow field names and types from SECRETS.DB.
; Borrow security information from SECINFO.DB.
tbl = CREATE "MySecrts.db"
        LIKE "Secrets.db"
        SECSTRUCT "SecInfo.db"
      ENDCREATE
endMethod
```

Privilege values and names for enumSecStruct

The following table lists numeric values and symbolic names for table and field privileges.

Value	Name	Description
0	None	Specifies no privileges
1	ReadOnly	Specifies a read-only field or table
3	Modify	Specifies a read and modify field or table
7	Insert	Specifies insert + all of the above privileges (table only)
15	InsDel	Specifies delete + all of the above privileges (table only)
31	Full	Specifies full rights (table only)
255	Unknown	Specifies privileges unknown

Family rights values and names for enumSecStruct

The following table lists numeric values and symbolic names for family rights.

Value	Name	Description
0	NoFamRights	Specifies no family rights
1	FormRights	Specifies the right to change forms only
2	RptRights	Specifies the right to change reports only
4	ValRights	Specifies the right to change val checks only
8	SetRights	Specifies the right to change image settings
15	AllFamRights	Specifies all of the above

enumTableProperties method

Writes the properties of a TCursor to a Corel Paradox table.

Syntax

```
enumTableProperties ( const tableName String ) Logical
```

Description

enumTableProperties writes the properties of a table associated with a TCursor to the table specified in *tableName*. If *tableName* already exists, this method prompts the user for confirmation before overwriting the table. If *tableName* is open, this method fails. You can also include an alias or path in *tableName*. If an alias or path is not specified, Corel Paradox creates the table in the working directory.

The following table displays the structure of *tableName*:

Field name	Field type	Description
TableName	A32	Specifies the table name only (i.e., no path, no extension)
PropertyName	A64	Specifies the property name (e.g., for Corel Paradox and dBASE tables: Name, Type, FieldCount, LogicalRecordSize, PhysicalRecordSize, KeySize, IndexCount, ValCheckCount, ReferentialCount, BookMarkSize, StableBookMarks, OpenMode, ShareMode)
PropertyValue	A255	Specifies the corresponding property value

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCENUMFIELDNAMES;',0,"Defaultoverview",)} **Related Topics**

enumTableProperties example

The following example uses **enumTableProperties** to write ORDERS.DB properties to ORDPROPS.DB. If ORDPROPS.DB exists, this code asks for confirmation before overwriting the table:

```
; showTblProps::pushButton
method pushButton(var eventInfo Event)
var
  tblName, propTbl String
  tc TCursor
  tv TableView
endVar
tblName = "Orders.db"
propTbl = "OrdProps.db"


if tc.open(tblName) then
  if isTable(propTbl) then
    if msgYesNoCancel("Confirm",
       propTbl + " exists. Overwrite it?") <> "Yes" then
      return
    endIf
  endIf
  ; Write Orders.db properties to OrdProps.db.
  tc.enumTableProperties(propTbl)
  ; Open newly created OrdProps.db table.
  tv.open(propTbl)
else
  msgStop("Stop!", "Can't open " + tblName + " table.")
endIf


endMethod
```

eot method

Determines whether a command attempts to move past the table's last record.

Syntax

```
eot ( ) Logical
```

Description

eot returns True if a command attempts to move past the table's last record; otherwise, it returns False. **eot** is reset by the next move operation.

eot (and **bot**) returns True if a command forces the TCursor to point to a nonexistent record. For example, assume that the *Customer* table has values in the first key field that range from 1,000 to 10,000. If you call **setRange** and point the TCursor to key values from 1 to 10 (outside the possible range of *Customer* values), the TCursor points to a nonexistent record. The following code fragment demonstrates how **setRange** can affect **eot** and **bot**:

```
var tc TCursor endvar
tc.open("Customer.db")
; Suppose values in field 1 range from 1,000 to 10,000.
tc.setRange(1, 10)           ; filter ranges from 1 to 10
  ; tc.eot() and tc.bot() are True at this point
```

If a call to **setGenFilter** forces the TCursor to point to a nonexistent record, **eot** and **bot** methods return True.

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCBOT;OPAL_METH_TCEND;OPAL_METH_TCHOME;',0,"Defaultoverview",)} Related Topics

eot example

In the following example, a **while** loop controls a TCursor's movement through the *Orders* table. When code within the loop attempts to move past the end of the table, **eot** returns True and the loop terminates.

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
  tc TCursor
  tblName String
  fldVal AnyType
endVar
tblName = "Customer.db"
if tc.open(tblName) then
  while tc.eot() = False      ; While subsequent commands do not
                              ; move past end of the table,
    message(tc."Customer No") ; display value in Customer No field,
    sleep(250)                ; pause for the message,
    tc.nextRecord()           ; move to the next record.
  endWhile
  msgInfo("End", "That's all, folks!")
else
  msgStop("Stop!", "Can't open " + tblName + " table.")
endIf
endMethod
```

familyRights method

Tests for a user’s ability to create or modify objects in a table’s family.

Syntax

```
familyRights ( const rights String ) Logical
```

Description

familyRights determines whether you can create or modify objects in a table's family. This method returns True if you have rights to the type of object specified in *rights*; otherwise, it returns False. *rights* is a single-letter string that indicates the object type to which you may have rights (e.g., F (form), R (report), S (image settings), or V (validity checks)). This method preserves the functionality required by Corel Paradox 3.5 tables but does not apply to tables created in versions of Corel Paradox after 3.5.

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCTRIG;',0,"Defaultoverview",)} Related Topics

familyRights example

The following example determines whether you have F rights to CUSTOMER.DB:

```
; showFRights::pushButton
method pushButton(var eventInfo Event)
var
  custTC TCursor
endVar


custTC.open("Customer.db")
msgInfo("Rights", "Form Rights: " + String(custTC.familyRights("F")))
; Displays True if you have Form rights to CUSTOMER.DB.


endMethod
```

fieldName method

Returns the name of a field.

Syntax

```
fieldName ( const fieldNum SmallInt ) String
```

Description

fieldName returns the name of field specified by *fieldNum*. Fields are numbered from left to right, beginning with 1.

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TBFNO;OPAL_METH_TCFTYP;OPAL_METH_TCFVAL;',0,"Defaultoverview",)} **Related Topics**

fieldName example

The following example uses **fieldName** to display the name of field number two in the *Answer* table. This code is attached to the built-in **pushButton** method of a button:

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
  tc TCursor
  fldName, tblName String
  fldNum SmallInt
endVar
tblName = "Answer.db"


if tc.open(tblName) then
  fldName = tc.fieldName(2)       ; store name of field 2 in fldName
  msgInfo("Field Name",           ; display field 2 field name
          "Field name for field 2 is\n" + fldName)
else
  msgStop("Sorry", "Can't open " + tblName + " table.")
endIf


endMethod
```

fieldNo method

Returns the position of a field in a table.

Syntax

```
fieldNo ( const fieldName String ) SmallInt
```

Description

fieldNo returns the position of the field specified by *fieldName*. Fields are numbered from left to right, beginning with 1.

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCFVAL;',0,"Defaultoverview",)} **Related Topics**

fieldNo example

In the following example, code is attached to the **pushButton** method for *thisButton*. When you press *thisButton*, this code uses **fieldNo** to display the position of *Common Name* in the *BioLife* table:

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
  tc TCursor
  fldNum SmallInt
endVar


if tc.open("biolife.db") then
  fldNum = tc.fieldNo("Common Name")   ; store field number in fldNum
  msgInfo("Field Number",
          "Common Name field is\n field number " + String(fldNum))
else
  msgInfo("Sorry", "Can't open BioLife.db table.")
endIf


endMethod
```

fieldRights method

Reports whether a user can read or modify a field in a table.

Syntax

```
1. fieldRights ( const fieldName String, const rights String ) Logical
2. fieldRights ( const fieldNum SmallInt, const rights String ) Logical
```

Description

fieldRights returns True if the user has *rights* to the field specified in *fieldName* or *fieldNum*; otherwise, it returns False. The value of *rights* must be an expression that evaluates to one of the following strings: ReadAll, ReadOnly, or None. Rights are obtained using **addPassword** (Session type). Rights cannot be acquired after the table is opened.

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCTRIG;',0,"Defaultoverview",)} Related Topics

fieldRights example

The following example uses **fieldRights** to determine whether a TCursor has adequate field rights before modifying the field's value:

```
; updateCust::pushButton
method pushButton(var eventInfo Event)
var
  custTC TCursor
endVar
custTC.open("Customer.db")
if custTC.locate("Name", "Unisco") then
  ; if we don't have sufficient rights to change the Name field
  if NOT custTC.fieldRights("Name", "ReadWrite") then
    ; display error message and abort operation
    msgStop("Error!", "Insufficient rights to change Name field")
  else
    ; otherwise, we have rights to make changes to the field
    custTC.edit()
    custTC.Name = "Unisco Worldwide, Inc."
    message("Changed Unisco to Unisco Worldwide, Inc.")
    custTC.endEdit()
  endIf
else
  msgStop("Error", "Can't find Unisco")
endIf

endMethod
```

fieldSize method

Returns the size of a field.

Syntax

```
1. fieldSize ( const fieldName String ) SmallInt
2. fieldSize ( const fieldNum SmallInt ) SmallInt
```

Description

fieldSize returns the size of a field, as defined when the table was created. The return value represents the maximum number of characters a field can contain. For example, given a field defined as Alpha20, **fieldSize** returns a value of 20. The return value can represent the maximum amount of data the field can display. For example, given a table or a Memo field **fieldSize** returns the number of characters that can be displayed.

Numeric fields in dBASE tables can specify the number of digits to display on each side of the decimal point. For example, a field defined as Number 8.2 displays up to 8 digits total, with 6 digits to the left of the decimal and 2 digits to the right. **fieldSize** returns the number of digits to the left of the decimal. To get the second part, use **fieldUnits2**.

For field types that do not display characters or numbers (e.g., OLE, binary, graphic), this method returns 0.

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCFNAM;OPAL_METH_TCFTYP;OPAL_METH_TCFUNIT;',0, "Defaultoverview",)} **Related Topics**

fieldSize example

The following example uses a dynamic array to store the size of each field in the *BioLife* table and displays the contents of the dynamic array in a dialog box:

```
; showFldSizes::pushButton
method pushButton(var eventInfo Event)
var
  tc TCursor
  i SmallInt
  fldSizes DynArray[] AnyType
  tblName String
endVar
tblName = "BioLife.db"


if tc.open(tblName) then
   ; this FOR loop loads the DynArray with BioLife.db field sizes
  for i from 1 to tc.nFields()
    fldSizes[tc.fieldName(i)] = tc.fieldSize(i)
  endFor
   ; now show the contents of the DynArray
  fldSizes.view(tblName + " field sizes.")
else
  msgStop("Sorry", "Can't open " + tblName + " table.")
endIf
endMethod
```

fieldType method

Returns the data type of a field.

Syntax

```
1. fieldType ( const fieldName String ) String
2. fieldType ( const fieldNum SmallInt ) String
```

Description

fieldType returns the data type of a field. If the specified field is not found, this method returns "unknown." The following tables list the possible return values for Corel Paradox and dBASE tables:

Corel Paradox Field Type	Return Value
Alpha	ALPHA
Autoincrement	AUTOINCREMENT
BCD	BCD
Binary	BINARY
Bytes	BYTES
Date	DATE
Formatted Memo	FMTMEMO
Graphic	GRAPHIC
Logical	LOGICAL
Long Integer	LONG
Memo	MEMO
Money	MONEY
Number	NUMBER
OLE	OLE
Short	SHORT
Time	TIME
Timestamp	TIMESTAMP

dBASE Field Type	Return Value
BINARY	BINARY
CHARACTER	CHARACTER
DATE	DATE
FLOAT	FLOAT
LOGICAL	LOGICAL
MEMO	MEMO
NUMBER	NUMERIC
OLE	OLE

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCFNO;',0,"Defaultoverview",)} Related Topics

fieldType example

The following example uses a dynamic array to store the data type of each field in the *BioLife* table and displays the contents of the dynamic array in a dialog box:

```
; showFldTypes::pushButton
method pushButton(var eventInfo Event)
var
  tc TCursor
  i SmallInt
  fldTypes DynArray[] AnyType
  tblName String
endVar
tblName = "BioLife.db"


if tc.open(tblName) then
   ; this FOR loop loads the DynArray with BioLife.db field types
  for i from 1 to tc.nFields()
    fldTypes[tc.fieldName(i)] = tc.fieldtype(i)
  endFor
   ; now show the contents of the DynArray
  fldTypes.view(tblName + " field types.")
else
  msgStop("Sorry", "Can't open " + tblName + " table.")
endIf
endMethod
```

fieldUnits2 method

Returns the number of decimal places defined for a numeric field in a dBASE table.

Syntax

```
1. fieldUnits2 ( const fieldName String ) SmallInt
2. fieldUnits2 ( const fieldNum SmallInt ) SmallInt
```

Description

fieldUnits2 returns the number of decimal places defined for a numeric field in a dBASE table. For a Corel Paradox table or any other driver or field type that does not require field units to be specified, this method returns 0.

Numeric fields in dBASE tables can specify the number of digits to display on each side of the decimal point. For example, a field defined as Number 8.2 displays up to 8 digits total, with 6 digits to the left of the decimal and 2 digits to the right. **fieldSize** returns the number of digits to the left of the decimal. To get the second part, use **fieldUnits2**.

For field types that do not display characters or numbers (e.g., OLE, binary, graphic), this method returns 0.

Example

 Related Topics

fieldUnits2 example

For the following example, the **pushButton** method for *thisButton* concatenates values returned from **fieldSize** and **fieldUnits2** so that both sides of the decimal point are expressed in a single number. For example, if a field's size is 11 and is defined with 2 decimal places, this method concatenates the values to 11.2. This code uses a DynArray to store concatenated values for each field in SCORES.DBF then displays the contents of the array in a dialog box:

```
; showFldSizes::pushButton
method pushButton(var eventInfo Event)
var
  tc TCursor
  i SmallInt
  fldSizes DynArray[] AnyType
  tblName String
  totalSize Number
endVar
tblName = "Scores.dbf"


if tc.open(tblName) then
   ; This FOR loop loads the DynArray with the full field spec.
   ; For example if fieldSize(1) = 11 and fieldUnits2(1) = 2,
   ; one value in the DynArray would be 11.2
  for i from 1 to tc.nFields()
    totalSize = numVal(String(tc.fieldsize(i)) + "." +
                       String(tc.fieldUnits2(i)))
    fldSizes[tc.fieldName(i)] = totalSize
  endFor
   ; now show the contents of the DynArray
  fldSizes.view(tblName + " total field sizes.")
else
  msgStop("Sorry", "Can't open " + tblName + " table.")
endIf
endMethod
```

fieldValue method

Reads the value of a specified field.

Syntax

```
1. fieldValue ( const fieldName String, var result AnyType ) Logical
2. fieldValue ( const fieldNum SmallInt, var result AnyType ) Logical
```

Description

fieldValue retrieves the value the field (*fieldName* or *fieldNum*) and assigns it to the variable *result*. This method returns True if successful; otherwise, it returns False.

You can also read the value of a specified field using dot notation. For example, the following statement uses dot notation to assign the *myPrice* variable with data from the Last Bid field:

```
myCost = tcVar."Last Bid"
```

The following statement uses **fieldValue** to achieve the same results:

```
tcVar.fieldValue("Last Bid", myCost)
```

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCSFVA;',0,"Defaultoverview",)} **Related Topics**

fieldValue example

The following example assumes that a form has at least one field, named *paymentField*. When you right-click *paymentField*, the code presents a PopUpMenu listing possible values for the field. When you choose a menu item from the list, that item is added to the field.

The following code is attached to the field's Var window:

```
; paymentField::Var
Var
  lkupTbl String
  menuArray Array[] String
  fldVal AnyType
  p1 PopUpMenu
  tc TCursor
endVar
```

The following code is attached to the field's **open** method. When the field opens, this code scans the *PayMethd* table and loads the *menuArray* array with values from the *Pay Method* field:

```
; paymentField::open
method open(var eventInfo Event)


lkupTbl = "PayMethd.db"
tc.open(lkupTbl)
scan tc :                              ; scan through table
  tc.fieldValue("Pay Method", fldVal) ; store field value in fldVal
  menuArray.addLast(fldVal)           ; add new element to menuArray
endScan
p1.addStaticText("Possible Values")   ; put static text at top of menu
p1.addSeparator()                     ; add a horizontal bar below static text
p1.addArray(menuArray)                ; add array to the menu


endMethod
```

The following code is attached to the field's **mouseRightUp** method. When you right-click the field, this code presents a PopUpMenu. The values that you choose is displayed in the field.

```
; paymentField::mouseRightUp
method mouseRightUp(var eventInfo MouseEvent)


disableDefault              ; don't show the default menu
choice = p1.show()          ; show the pop-up menu
if NOT isBlank(choice) then ; if user did not press Esc
  self.value = choice       ; enter choice into the field
endIf


endMethod
```

forceRefresh method

Forces a TCursor to point to the data in the underlying table.

Syntax

forceRefresh( ) Logical

Description

forceRefresh empties a TCursor’s record buffer and refreshes it with data from the underlying table. The record position is maintained, provided the record still exists in the table. On an SQL server, a call to **forceRefresh** forces a read from the server. This is the only way to get a refresh from the server; it may be a time-consuming operation. **forceRefresh** only works on SQL tables that have a unique index.

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCCURE;OPAL_METH_UIFORCEREFRESH;',0,"Defaultoverview",)} Related Topics

forceRefresh example

The following example opens a TCursor on the Orders table and executes two scan loops to perform two calculations. The first calculation returns the total quantity of orders from California. The code then calls **forceRefresh** to get the latest data from the table before executing the second scan loop. The second calculation calculates the total quantity of orders from Florida.

```
method pushButton(var eventInfo Event)
   var
      tc       TCursor
      tName,
      fName,
      fVal_1,
      fVal_2   String
      caQty,
      flQty    LongInt
   endVar


   ; initialize variables
   tName  = "orders" ; assign table name
   fName  = "State"  ; assign field name
   caQty  = 0        ; assign CA quantity
   flQty  = 0        ; assign FL quantity
   fVal_1 = "CA"     ; assign 1st field value
   fVal_2 = "FL"     ; assign 2nd field value


   tc.open(tName)


   scan tc for tc.State = fVal_1:
      caQty = caQty + tc.Qty
   endScan


; during the first scan, other users may
; change data in the underlying table


   tc.forceRefresh() ; Get latest data from table


   scan tc for tc.State = fVal_2:
      flQty = flQty + tc.Qty
   endScan


   msgInfo("CA Qty and FL Qty:",
           "CA = " + String(caQty) + "\n" + "FL = " + String(flQty))


endMethod
```

getGenFilter method

Retrieves the filter criteria associated with a TCursor.

Syntax

```
1. getGenFilter ( criteria DynArray[ ] AnyType ) Logical

2. getGenFilter ( criteria Array[ ] AnyType [ , fieldName Array[ ] AnyType ] ) Logical

3. getGenFilter ( criteria String ) Logical
```

Description

getGenFilter retrieves the filter criteria associated with a TCursor. This method assigns values to a dynamic array (DynArray) variable in Syntax 1, or to two Array variables that you declare and include as arguments in Syntax 2.

In Syntax 1, the DynArray *criteria* lists fields and filtering conditions as follows: the index is the field name or number (depending on how it was set), and the item is the corresponding filter expression.

In Syntax 2, the Array *criteria* lists filtering conditions, and the optional Array *fieldName* lists corresponding field names. If you omit *fieldName*, conditions apply to fields in the order they appear in the *criteria* array (the first condition applies to the first field in the table, the second condition applies to the second field, and so on).

If the arrays used in Syntax 2 are resizeable, this method sets the array size to equal the number of fields in the underlying table. If fixed-size arrays are used, this method stores as many criteria as possible, beginning with criteria field 1. If there are more array items than fields, the remaining items are empty. If there are more fields than items, this method fills the array.

In Syntax 3, filter criteria is assigned to a String variable *criteria* that you must declare and pass as an argument.

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCSETGENFILTER;OPAL_METH_TCDROPGENFILTER;OPAL_METH_TCSETRANGE;OPAL_METH_UIGETGENFILTER;OPAL_INFO_TBUSINGRANGESANDFILTERS;',0,"Defaultoverview",)} **Related Topics**

getGenFilter example

In the following example, the **pushButton** method for a button named *btnShowFilter* uses **getGenFilter** to fill a dynamic array (DynArray) named *dyn* with a TCursor's filter criteria. The code then determines whether the current criteria filters the State/Prov field with a value of CA, and resets the filter if necessary.

```
;btnShowFilter :: pushButton
method pushButton(var eventInfo Event)
var
      custTC      TCursor
      dyn         DynArray[] AnyType
      keysAr      Array[] AnyType
stFilterFld,
stCriteria   String
endVar
stFilterFld = "State/Prov"
stCriteria  = "CA"
custTC.open("Customer")


custTC.getGenFilter(dyn)   ; Get filter info.
dyn.getKeys(keysAr)
if keysAr.contains(stFilterFld) then
if dyn[stFilterFld] = stCriteria then
return               ; Filter is set correctly.
endIf
else
dyn.empty()             ; Set filter criteria correctly.
dyn[stFilterFld] = stCriteria
custTC.setGenFilter(dyn)
endIf
endMethod
```

getIndexName method

Retrieves the name of a table's current index.

Syntax

1. (Corel Paradox tables) **getIndexName (*indexName* String)** Logical
2. (dBASE tables) **getIndexName (*indexName* String [, *tagName* String])** Logical

Description

getIndexName retrieves the name of the current index. **getIndexName** can also retrieve the current tag for dBASE tables. This method assigns values to String variables that you must declare and provide as arguments.

For more information on indexes, see About keys and indexes in tables in the User's Guide help.

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCGETGENFILTER;OPAL_METH_TCSETGENFILTER;',0,"Defaultoverview",)} Related Topics

getIndexName example

The following example retrieves and displays the name of the index associated with the *Orders* table:

```
method pushButton(var eventInfo Event)
   var
      ordersTC TCursor
      indexName String
   endVar


   ordersTC.open("orders")


   ; Get the index name and assign the value to the String variable indexName.
   ordersTC.getIndexName(indexName)


   if indexName.isAssigned() then
      indexName.view("Current index")
   else
      msgInfo("indexName", "No value for indexName.")
   endIf
endMethod
```

getLanguageDriver method

Returns the name of the table's current language.

Syntax

```
getLanguageDriver ( ) String
```

Description

getLanguageDriver returns a String value that specifies the language driver for a table.

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCGETLANGUAGEDRIVERDESC;OPAL_METH_QUSETLANGUAGEDRIVER;',0,"Defaultoverview",)} Related Topics

getLanguageDriver example

The following example displays the language driver for the *Customer* table in a dialog box:

```
; getDriver::pushButton
method pushButton(var eventInfo Event)
var
  tc TCursor
endVar
tc.open("Customer.db")
msgInfo("", tc.getLanguageDriver())  ; displays "ascii"
endMethod
```

getLanguageDriverDesc method

Returns the name of the table's current language driver description.

Syntax

```
getLanguageDriverDesc ( ) String
```

Description

getLanguageDriverDesc returns a String value that specifies the table's language driver.

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCGETLANGUAGEDRIVER;OPAL_METH_QUSETLANGUAGEDRIVER;',0,"Defaultoverview",)} Related Topics

getLanguageDriverDesc example

The following example displays the language driver description for the *Customer* table in a dialog box:

```
; getDriverDesc::pushButton
method pushButton(var eventInfo Event)
var
  tc TCursor
endVar
tc.open("Customer.db")
msgInfo("", tc.getLanguageDriverDesc())  ; displays "Corel Paradox ascii"
endMethod
```

getRange method

Retrieves the values that specify a range for a TCursor.

Syntax

```
getRange ( var rangeVals Array[ ] String ) Logical
```

Description

getRange retrieves the values that specify a range for a TCursor. This method assigns values to an Array variable that you declare and include as an argument. The following table displays the array values and the corresponding range criteria:

Number of array items	Range specification
No items (empty array)	Specifies no range criteria is associated with the Table variable
One item	Specifies a value for an exact match on the first field of the index
Two items	Specifies a range for the first field of the index
Three items	The first item specifies an exact match for the first field of the index; items 2 and 3 specify a range for the second field of the index.
More than three items	For an array of size *n*, specifies exact matches on the first *n*-2 fields of the index. The last two array items specify a range for the *n*-1 field of the index

If the array is resizeable, this method sets the array size to equal the number of fields in the underlying table. If fixed-size arrays are used, this method stores as many criteria as it can, starting with criteria field 1. If there are more array items than fields, the remaining items are left empty; if there are more fields than items, this method fills the array and then stops.

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCSETRANGE;OPAL_METH_TCGETGENFILTER;OPAL_METH_TBGETRANGE;OPAL_METH_UIGETRANGE;OPAL_INFO_TBUSINGRANGESANDFILTERS;',0,"Defaultoverview",)} **Related Topics**

getRange example

In the following example, a button on a form is used to display the number of orders for any customer number per month. Assume that a form with the *Orders* table in its data model contains a Customer_No field, a Month field, and a button named *btnCustOrdersByMonth*. In this example a secondary index named *secCustomerMonth,* **getRange**, **getIndexName**, **switchIndex** and **setRange** is used to speed up the task.

```
;btnCustOrdersByMonth :: pushButton
method pushButton(var eventInfo Event)
var
      tc               TCursor
      nuCustomer   Number
arGet,
arSet         Array[2] AnyType
stMonth,
stActiveInd,
stDisplay   String
endVar
   nuCustomer = Customer_No.value   ;Customer field on form.
   nuCustomer.view("Customer #:")   ;Allow user to alter cust #.


   stMonth = Month.value       ;Month field on form.
   stMonth.view("Month:")      ;Allow user to alter month.


   arSet[1] = nuCustomer       ;Set array to range criteria.
arSet[2] = stMonth
   tc.attach(Customer_No)      ;Attach tc to Customer field.


   tc.getIndexName(stActiveInd)    ;Get the active index name.
if stActiveInd = "secCustomerMonth" then
      tc.getRange(arGet)          ;Get the current range.
      if arGet <> arSet then      ;Compare current range.
tc.setRange(nuCustomer, stMonth, stMonth)
endIf
else
;You must create a secondary index named secCustomerMonth
;for this example to work.
tc.switchIndex("secCustomerMonth")
tc.setRange(nuCustomer, stMonth, stMonth)
endIf
stDisplay = String(nuCustomer) + " had "
•   String(tc.nRecords()) +
" orders in " + stMonth
msgInfo("Orders in a month", stDisplay)
endMethod
```

handle method

Returns a cursor handle for use in an external DLL call.

Syntax

```
handle ( ) LongInt
```

Description

handle returns the cursor handle for use in an external DLL call.

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_AFATTACH;OPAL_METH_AFWINDOWHANDLE;',0,"Default overview",)} Related Topics

handle example

In the following example, the displayTable method of the PDXTEST.DLL is called with the handle of the TCursor. The following code appears in an ObjectPAL Editor window for the script's built-in **run** method"

```
; Define the prototype information for the displayTable method
; of PDXTEST.DLL
Uses PDXTEST
     displayTable( handle CLONG ) CLONG
endUses

method run(var eventInfo Event)
var
    tc   TCursor
    hCur LongInt
endvar

; Open the TCursor and get the handle of the opened table
tc.open( "aspace.db" )
hCur = tc.handle()

; Call the DLL's displayTable method, which displays the table's data.
; The DLL method should clone the cursor then close the cloned cursor
; after it has completed the pack.
displayTable ( hCur )

; Close the TCursor
tc.close()
endMethod
```

home method

Moves to a table's first record.

Syntax

home ( ) Logical

Description

home moves to a table's first record.

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCEND;OPAL_METH_TCMTRE;OPAL_METH_TCMTRENO;OPAL_METH_TCNERE;OPAL_METH_TCPRRE;OPAL_METH_TCSKIP;',0,"Defaultoverview",)} Related Topics

home example

For the following example, the **pushButton** method associates a *TCursor* with the *Orders* table and loads an array with field values in a **scan** loop. When the loop terminates, the TCursor is positioned in the table's last record. This code uses **home** to move the TCursor back to the table's first record:

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
  tc TCursor
  fldArray Array[] AnyType
  fldVal AnyType
endVar
tc.open("Orders.db")
fldArray.grow(tc.nRecords())
; scan table and store order numbers in fldArray
scan tc:
  tc.fieldValue(1, fldVal)
  fldArray[tc.recNo()] = tc.fldVal
endScan
; TCursor is on the last record after the scan loop

fldArray.view()            ; display contents of array


tc.home()                  ; move TCursor to the first record
endMethod
```

initRecord method

Empties the record buffer.

Syntax

```
initRecord ( ) Logical
```

Description

initRecord initializes the record buffer by filling it with blanks (*not* spaces). If you have set default values for fields, **initRecord** initializes those fields with the default.

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCCREC;OPAL_METH_TCCURE;',0,"Defaultoverview",)}

Related Topics

initRecord example

See the example for lockRecord.

insertAfterRecord method

Inserts a record below the active record.

Syntax

```
insertAfterRecord ( [ const pointer TCursor ] ) Logical
```

Description

insertAfterRecord inserts a record below active record. This method can be used to add new records to the end of a table. The optional argument *pointer* inserts the record pointed to by a different TCursor. Omitting the argument inserts a blank record.

If the table is indexed, the record is placed in its sorted position when the data is committed; otherwise, it is inserted after the active record.

This method fails if the table is not in Edit mode, or if the active record cannot be committed (e.g., because of a key violation).

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCIREC;OPAL_METH_TCIBER;',0,"Defaultoverview",)}

Related Topics

insertAfterRecord example

The following example assumes that a form has a table frame named *CUSTOMER* that is bound to CUSTOMER.DB. When the user deletes a record, the built-in action method for *CUSTOMER* moves the record to CUSTARC.DB before deleting it from *CUSTOMER*.

You could use **copyFromArray** and **copyToArray** to accomplish the same thing, but if you use **insertAfterRecord** you don't have to store the record in an array in order to copy it.

This code uses the optional argument *pointer* to insert the record pointed to by a TCursor:

```
; CUSTOMER::action
method action(var eventInfo ActionEvent)
var
  tcCust, tcArc TCursor
endVar
if eventInfo.id() = DataDeleteRecord then ; if user attempts to delete a record
  if thisForm.Editing = True then         ; if form is in Edit mode
    disableDefault                        ; don't process DataDeleteRecord yet


    if msgYesNoCancel("Confirm",          ; if user confirms delete
       "Delete the active record?") = "Yes" then
      tcCust.attach(CUSTOMER)             ; sync TCursor to CUSTOMER pointer
      if tcArc.open("CustArc.db") then
        tcArc.edit()
        tcArc.end()                       ; move to end of table
        tcArc.insertAfterRecord(tcCust)   ; insert current CUSTOMER record
                                          ; after last record in CustArc.db
        doDefault                         ; process DataDeleteRecord now
      else
        msgStop("Stop!", "Sorry, Can't archive record.")
      endIf
    else                                  ; else user didn't confirm delete
      message("Record not deleted.")
    endIf
  else                                    ; else form is not in Edit mode
    msgStop("Stop!", "Press F9 to edit data.")
  endIf
endIf
endMethod
```

insertBeforeRecord method

Inserts a record above the active record.

Syntax

```
insertBeforeRecord ( [ const pointer TCursor ] ) Logical
```

Description

insertBeforeRecord inserts a record above the active record. You can use the optional argument *pointer* to insert the record pointed to by another TCursor. If you omit the *pointer* argument, a blank record is inserted.

If the table is indexed, the record is placed in its sorted position when the data is committed; otherwise, it is inserted after the active record.

This method fails if the table is not in Edit mode, or if the active record cannot be committed (e.g., because of a key violation).

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCIREC;OPAL_METH_TCIAFR;',0,"Defaultoverview",)}

Related Topics

insertBeforeRecord example

The following example assumes that a form has a table frame named *CUSTOMER* that is bound to CUSTOMER.DB. When the user deletes a record, the built-in action method for *CUSTOMER* moves the record to CUSTARC.DB before deleting it from *CUSTOMER*.

You could use **copyFromArray** and **copyToArray** to accomplish the same thing, but if you use **insertAfterRecord** you don't have to store the record in an array in order to copy it.

This code uses the optional argument *pointer* to insert the record pointed to by a TCursor:

```
; CUSTOMER::action
method action(var eventInfo ActionEvent)
var
  tcCust, tcArc TCursor
endVar
if eventInfo.id() = DataDeleteRecord then ; if user attempts to delete a record
  if thisForm.Editing = True then         ; if form is in Edit mode
    disableDefault                        ; don't process DataDeleteRecord yet


    if msgYesNoCancel("Confirm",          ; if user confirms delete
       "Delete the active record?") = "Yes" then
      tcCust.attach(CUSTOMER)             ; sync TCursor to CUSTOMER pointer
      if tcArc.open("CustArc.db") then
        tcArc.edit()
        tcArc.insertBeforeRecord(tcCust)  ; insert current CUSTOMER record
                                          ; before active record in CustArc.db
        doDefault                         ; process DataDeleteRecord now
      else
        msgStop("Stop!", "Sorry, Can't archive record.")
      endIf
    else                                  ; else user didn't confirm delete
      message("Record not deleted.")
    endIf
  else                                    ; else form is not in Edit mode
    msgStop("Stop!", "Press F9 to edit data.")
  endIf
endIf
endMethod
```

insertRecord method

Inserts a record above the active record.

Syntax

```
insertRecord ( [ const pointer TCursor ] ) Logical
```

Description

insertRecord inserts a record into a table above the active record. You can use the optional argument *pointer* to insert the record pointed to by another TCursor. If you omit the *pointer* argument, a blank record is inserted.

If the table is indexed, the record is placed in its sorted position when the data is committed; otherwise, it is inserted after the active record.

This method fails if the table is not in Edit mode, or if the active record cannot be committed (e.g., because of a key violation).

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCIAFR;OPAL_METH_TCIBER;',0,"Defaultoverview",)}
Related Topics

insertRecord example

The following example assumes that a form has a table frame named *CUSTOMER* that is bound to CUSTOMER.DB. When the user deletes a record, the built-in action method for *CUSTOMER* moves the record to CUSTARC.DB before deleting it from *CUSTOMER*.

You could use **copyFromArray** and **copyToArray** to accomplish the same thing, but if you use **insertAfterRecord** you don't have to store the record in an array in order to copy it.

This code uses the optional argument *pointer* to insert the record pointed to by a TCursor:

```
; CUSTOMER::action
method action(var eventInfo ActionEvent)
var
  tcCust, tcArc TCursor
endVar
if eventInfo.id() = DataDeleteRecord then ; if user attempts to delete a record
  if thisForm.Editing = True then         ; if form is in Edit mode
    disableDefault                        ; don't process DataDeleteRecord yet


    if msgYesNoCancel("Confirm",          ; if user confirms delete
       "Delete the active record?") = "Yes" then
      tcCust.attach(CUSTOMER)             ; sync TCursor to CUSTOMER pointer
      if tcArc.open("CustArc.db") then
        tcArc.edit()
        tcArc.insertRecord(tcCust)        ; insert current CUSTOMER record
                                          ; before active record in CustArc.db
        doDefault                         ; process DataDeleteRecord now
      else
        msgStop("Stop!", "Sorry, Can't archive record.")
      endIf
    else                                  ; else user didn't confirm delete
      message("Record not deleted.")
    endIf
  else                                    ; else form is not in Edit mode
    msgStop("Stop!", "Press F9 to edit data.")
  endIf
endIf
endMethod
```

instantiateView method

Copies an in-memory TCursor to a physical table and points the TCursor to it.

Syntax

```
1. instantiateView ( const tableName String ) Logical
2. instantiateView ( const tableVar Table ) Logical
```

Description

instantiateView copies an in-memory TCursor to a physical table and points the TCursor to it. This method returns True if successful; otherwise, it returns False.

Syntax 1 creates the table using the name specified in *tableName*.

Syntax 2 associates the table with the Table variable specified in *tableVar*.

Use this method after executing a query that generates a TCursor onto a live query view. **instantiateView** copies the data from the live query view to a table on disk and makes the TCursor point to it. You can use the TCursor to manipulate the table's data. The resulting table has no relationship to the underlying tables in the query.

For more information on live query views, see Live query views in the User's Guide help.

You can also use **instantiateView** with TCursors created by ObjectPAL methods.

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCISINMEMORYTCURSOR;OPAL_METH_TCISVIEW;OPAL_METH_QUWANTINMEMORYTCURSOR;',0,"Defaultoverview",)} Related Topics

instantiateView example

The following example executes a query to a TCursor and determines whether the result is a live query view. If so, the code calls **instantiateView** to write the view to a physical table. The table is displayed in a Table window.

```
method pushButton(var eventInfo Event)
const
      kName = "salary"
endConst
var
      qbeVar     Query
      tcAnswer   TCursor
      tvAnswer   TableView
endVar


qbeVar.readFromFile(kName)
qbeVar.executeQBE(tcAnswer)

if tcAnswer.isView() then
     tcAnswer.instantiateView(kName)
     tvAnswer.open(kName)
else
     return
endIf
endMethod
```

isAssigned method

Reports whether a TCursor variable has been assigned a value.

Syntax

```
isAssigned ( ) Logical
```

Description

isAssigned returns True if a TCursor variable has been assigned a value using open or attach; otherwise, it returns False.

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCOPEN;OPAL_METH_TCCLO;',0,"Defaultoverview",)}

Related Topics

isAssigned example

The following example associates a TCursor with a table, displays the last record and closes the TCursor. The code displays a message indicating whether the TCursor variable remains assigned when the TCursor is closed. This code is attached to the built-in **pushButton** method for *thisButton*:

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
  tc TCursor
endVar
tc.open("Orders.db")         ; open a TCursor for Orders.db
tc.end()                     ; move to end of the table


; display information in last record
msgInfo("Last Order", "Order number: " +
String(tc."Order No") + " \nOrder date: " + String(tc."Sale Date"))


tc.close()                   ; attempt to close TCursor


; if close is successful, this displays False (tc is no longer assigned)
; otherwise, it displays True (tc is still assigned if close fails)
msgInfo("Is tc Assigned?", tc.isAssigned())


endMethod
```

isEdit method

Reports whether a TCursor is in Edit mode.

Syntax

isEdit ( ) Logical

Description

isEdit returns True if the TCursor is in Edit mode; otherwise, it returns False. If you attach a TCursor to a display manager that is in Edit mode (e.g., a UIObject or TableView), the TCursor will be in Edit mode as well.

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCCOED;OPAL_METH_TCENDED;',0,"Defaultoverview",)}

Related Topics

isEdit example

The following example assumes that a form has a button and a table frame that is bound to the *Customer* table. The **pushButton** method for *thisButton* attaches a TCursor to the table frame and uses **isEdit** to determine whether the TCursor is in Edit mode. If the table frame is in Edit mode when the TCursor is attached, the TCursor is also in Edit mode.

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
  tc TCursor
endvar


; attach to the table frame
tc.attach(CUSTOMER)


; if CUSTOMER was in Edit mode, tc will be in Edit mode too


if NOT tc.isEdit() then    ; test whether tc is in Edit mode
  tc.edit()
endIf


if tc.locate("Name", "Action Club") then
  tc.phone = "808-555-1234"
else
  msgStop("Sorry", "Can't find Action club")
endIf


endMethod
```

isEmpty method

Determines whether a table contains any records.

Syntax

```
isEmpty ( ) Logical
```

Description

isEmpty returns True if there are no records in the table associated with the TCursor; otherwise, it returns False.

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCEMPT;OPAL_METH_TCNREC;OPAL_METH_TCIREC;',0,"Defaultoverview",)} **Related Topics**

isEmpty example

In the following example the **pushButton** method for the *rptRecNo* button displays the number of records in the *Orders* table. If the table is empty, this code alerts the user that the table is empty:

```
; rptRecNo::pushButton
method pushButton(var eventInfo Event)
var
  tc TCursor
  tblName String
endVar
tblName = "Orders.db"


if tc.open(tblName) then
  if tc.isEmpty() then                          ; if Orders.db is empty
    msgStop("Hey!",
             tblName + " table is empty!")
  else
    msgInfo(tblName + " table has",             ; report number of records
            String(tc.nRecords()) + " records")
  endIf
else
  msgStop("Sorry", "Can't open " + tblName + " table.")
endIf
endMethod
```

isEncrypted method

Reports whether a table is password-protected.

Syntax

```
isEncrypted ( ) Logical
```

Description

isEncrypted returns True if a table is password-protected; otherwise, it returns False. You cannot open a TCursor on an encrypted table until you use **addPassword** (Session type) to present the required password. Use **tableRights** to report whether a user has access rights to the table.

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCTRIG;OPAL_METH_SSAPASS;OPAL_METH_TBPROT;',0,"Defaultoverview",)} Related Topics

isEncrypted example

The following example determines whether the *Customer* table is encrypted:

```
; thisButton::pushButton
method open(var eventInfo Event)
var
  tc TCursor
endvar


if tc.open("Customer.db") then
  if tc.isEncrypted() then
    msgInfo("Table is protected", "An acceptable password has been presented.")
  endif
else
  msgStop("Error", "Can't open the Customer table.")
endIf


endMethod
```

isInMemoryTCursor method

Reports whether a TCursor points to a table in system memory or to a physical table.

Syntax

isInMemoryTCursor ( ) Logical

Description

isInMemoryTCursor returns True if the TCursor is associated with a table in system memory (e.g., a table generated by an ObjectPAL method that enumerates information to a TCursor); otherwise, it returns False.

By default, when you execute a query, Corel Paradox attempts to create a live query view. Use **isInMemoryTCursor** to determine whether the query creates or an in-memory answer table. If the query creates a live query view, changes made to the TCursor affect the underlying tables. If the query creates an in-memory answer table, the underlying tables are not affected. If the query results in a live query view, **isInMemoryTCursor** returns False and **isView** returns True. You can use **wantInMemoryTCursor** to specify how to create a TCursor resulting from a query.

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_QUEXECUTEQBE;OPAL_METH_QUWANTINMEMORYTCURSOR;OPAL_METH_TCINSTANTIATEVIEW;OPAL_METH_TCISINMEMORYTCURSOR;OPAL_METH_TCISVIEW;',0,"Defaultoverview",)} Related Topics

isInMemoryTCursor example

The following example executes a query from a file and uses a **scan** loop to increase the salary of each employee by 12 percent. Because you cannot determine whether the query will create a live query view before it is run, this code calls **isInMemoryTCursor** to prevent changes from affecting the actual employee salary data:

```
method pushButton(var eventInfo Event)
   var
      qbeVar    Query
      tcAnswer  TCursor
   endVar


   ; Read the query from a file.
   qbeVar.readFromFile("Salary.qbe")


   ; We don't know if this query will generate a live
   ; query view, so use isInMemoryTCursor to find out.
   if qbeVar.executeQBE(tcAnswer) then


      ; If it is in memory (i.e., not live) and
      ; see the effects of a 12% raise for all employees.
      if tcAnswer.isInMemoryTCursor() then
         nuOldTotalPayroll = tcAnswer.cSum("Salary")


         tcAnswer.edit()
         scan tcAnswer :
            tcAnswer.Salary = tcAnswer.Salary * .15
         endScan
         tcAnswer.endEdit()


         nuNewTotalPayroll = tcAnswer.cSum("Salary")

         msgInfo("Before raise: " + String(nuOldTotalPayroll),
                 "After raise: " + String(nuNewTotalPayroll))


      else
         ; If it is live, inform user and quit the method.
         msgStop("Live query view",
             "Edits would affect the underlying table.")
         return
      endIf
   else
      errorShow()
   endIf


endmethod
```

isOnSQLServer method

Reports whether a TCursor is associated with a table on a SQL server.

Syntax

isOnSQLServer ( ) Logical

Description

isOnSQLServer returns True if the TCursor is associated with a table on a SQL server; otherwise, it returns False.

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCISOPENONUNIQINDX;',0,"Defaultoverview",)}

Related Topics

isOnSQLServer example

The following example is a custom method that uses **isOnSQLServer** to determine whether a TCursor is associated with a remote table. If **isOnSQLServer** returns True, this code displays a **msgQuestion** dialog box and prompts the user to confirm the lock on the remote table:

```
method confirmRemoteLock(const tc TCursor) Logical


   if tc.isOnSQLServer() then

      ; you might not want to lock remote tables
      if msgQuestion("Lock table?",
                     "Lock a remote table?") = "Yes" then
         return True
      else
         return False
      endIf
   endIf
endMethod
```

isOpenOnUniqueIndex method

Reports whether a TCursor is open on a unique index.

Syntax

```
isOpenOnUniqueIndex ( ) Logical
```

Description

isOpenOnUniqueIndex returns True if a TCursor is open on a unique index; otherwise, it returns False. A unique index is an index that does not allow duplicate key values.

This method allows you to update remote tables easily. Remote operations (e.g., editing data or deleting records) may fail unless the TCursor is opened on a unique index.

Example

isOpenOnUniqueIndex example

The following example is a custom method that calls **isOpenOnUniqueIndex** before placing the TCursor in Edit mode:

```
method editIfUniqueIndex(const tc TCursor) Logical
   if tc.isOpenOnUniqueIndex() then
      return tc.edit()
   else
      return False
   endIf
endMethod
```

isRecordDeleted method

Reports whether the active record has been deleted From a dBASE table.

Syntax

isRecordDeleted ( ) Logical

Description

isRecordDeleted reports whether the active record has been deleted. **isRecordDeleted** works only for dBASE tables because deleted Corel Paradox records can't be displayed. This method returns True if the active record has been deleted; otherwise, it returns False.

By default, deleted records in a dBASE table are not displayed. To display deleted records in the table, call **showDeleted**; otherwise, deleted records are not visible to **isRecordDeleted**.

 **Example**

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCISSHOWDELETEDON;OPAL_METH_TCSHDEL;',0,"Defaultoverview",)} **Related Topics**

isRecordDeleted example

The following example opens a TCursor for the SCORES.DBF dBASE table and uses **showDeleted** to display the table's deleted records. This code then attempts to locate a specific record in the table. This example uses **isRecordDeleted** to determine whether the record has been deleted. If it returns true, the record is undeleted using **undeleteRecord**. The following code is attached to the **pushButton** method for *thisButton*:

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
  tc TCursor
endVar
tc.open("Scores.dbf")                ; open TCursor on a dBASE table
tc.showDeleted()                     ; show deleted records
if tc.locate("Name", "Jones") then   ; if locate finds Jones in Name field
  if tc.isRecordDeleted() then       ; if the record has been deleted
    tc.edit()                        ; begin Edit mode
    tc.undeleteRecord()              ; undelete the record
    message("Jones record undeleted")
  endIf
else
  msgStop("Error", "Can't find Jones.")
endIf
endMethod
```

isShared method

Reports whether a table is currently shared with another user on the network.

Syntax

isShared ( ) Logical

Description

isShared returns True if another user has opened the table specified by a TCursor; otherwise, it returns False.

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCISASSIGNED;OPAL_METH_TCISVA;',0,"Defaultoverview",)} **Related Topics**

isShared example

In the following example, a form's built-in **open** method determines whether CUSTOMER.DB is currently shared by another user. If it is, the user is warned and given the option to continue or abort.

```
; thisPage::open
method open(var eventInfo Event)
var
  tc TCursor
endVar
tc.open("Customer.db")              ; open a TCursor for Customer
if tc.isShared() then               ; if table is currently shared
  if msgYesNoCancel("Continue?",    ; ask for confirmation
     "Customer table is currently being shared.\n" +
     "Continue anyway?") <> "Yes" then


    close()                         ; close this form
  endIf
endIf
endMethod
```

isShowDeletedOn method

Reports whether deleted records in a dBASE table are displayed.

Syntax

isShowDeletedOn ( ) Logical

Description

isShowDeletedOn reports whether the table pointed to by a TCursor displays its deleted records. Use the **showDeleted** method display deleted records and use **isShowDeletedOn** to determine states. **isShowDeletedOn** applies only to dBASE tables.

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCCOMP;OPAL_METH_TCISRECORDDELETED;OPAL_METH_TCSHDEL;OPAL_METH_TCSHDEL;',0,"Defaultoverview",)} **Related Topics**

isShowDeletedOn example

The following example calls **showDeleted** to display deleted records in ORDERS.DBF if **isShowDeletedOn** returns False:

```
; showDeletedRecs::pushButton
method pushButton(var eventInfo Event)
var
  dbfTC TCursor
endVar
if dbfTC.open("Orders.dbf") then
  if NOT dbfTC.isShowDeletedOn() then   ; if deleted records are not shown
    dbfTC.showDeleted(Yes)              ; show deleted records
  endIf
else
  msgStop("Sorry", "Can't open Orders.dbf table.")
endIf
endMethod
```

isValid method

Reports whether the contents of a field are valid and complete.

Syntax

```
1. isValid ( const fieldName String, const value AnyType ) Logical
2. isValid ( const fieldNum SmallInt, const value AnyType ) Logical
```

Description

isValid reports whether the value specified in *value* conforms with field type and validity checks for the field specified in *fieldNum* or *fieldName*. This method allows you to determine whether a new field value is valid before you attempt to post the record.

isValid returns True if *value* conforms to field type and validity checks; otherwise, it returns False.

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCPOSTRECORD;OPAL_METH_TCSFVA;',0,"Defaultoverview",)} Related Topics

isValid example

The following example uses **isValid** to determine whether a value is valid for a Date field. If the value is not valid, this code warns the user; otherwise the value is entered into the field. The following code is attached to the **pushButton** method for *thisButton*:

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
  tc TCursor
  tryValue String
endVar
tryValue = "100/5/1994" ; Invalid date.
tc.open("Orders.db")
if NOT tc.isValid("Sale Date", tryValue) then
  msgStop("Error",
  String(tryValue) + " is not valid for this field.")
else                   ; this condition is never met
  tc."Sale Date" = tryValue
  tc.postRecord()
endIf

endMethod
```

isView method

Reports whether a TCursor is associated with a live query view.

Syntax

```
isView ( ) Logical
```

Description

isView returns True if the TCursor is associated with a live query view; otherwise, it returns False.

If **isView** is True, **isInMemoryTCursor** returns False.

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCINSTANTIATEVIEW;OPAL_METH_TCISINMEMORYTCURSOR;OPAL_METH_QUWANTINMEMORYTCURSOR;',0,"Defaultoverview",)} Related Topics

isView example

See the **instantiateView** example.

locate method

Searches for a specified field value.

Syntax

```
1. locate ( const fieldName String, const exactMatch AnyType [ , const fieldName String, const exactMatch AnyType ] * ) Logical

2. locate ( const fieldNum SmallInt,  const exactMatch AnyType [ , const fieldNum SmallInt, const exactMatch AnyType ] * ) Logical
```

Description

locate searches a table for values that match the criteria specified in one or more field value pairs. Specify the value to search for in *exactMatch* and the field to search in *fieldName* or *fieldNum*. This method guarantees that the first value matching *exactMatch* is found and given the current view of the records. If the TCursor is using a secondary index, **locate** finds the first record in the secondary index order.

The search begins at the top of the table, but if no match is found, the TCursor returns to the original record. If a match is found, the TCursor moves to that record. This operation fails if the active record cannot be posted (e.g., because of a key violation).

Note

- The search is case-sensitive unless **ignoreCaseInLocate** (Session type) is set to True.

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCLNEX;OPAL_METH_TCLPAT;OPAL_METH_TCLNPA;',0,"Defaultoverview",)} Related Topics

locate example

In the following example, the **pushButton** method for the *fixSpelling* button searches for a value in the *Name* field of the *Customer* table. If **locate** is successful, this code replaces the name with a new value and informs the user of the change:

```
; fixSpelling::pushButton
method pushButton(var eventInfo Event)
var
  ordTC TCursor
endVar


ordTC.open("Customer.db")
; if locate finds "Proffessional Divers, Ltd." in the Name field
if ordTC.locate("Name", "Proffessional Divers, Ltd.") then
  ; begin Edit mode
  ordTC.edit()
  ; correct spelling (Professional)
  ordTC.Name = "Professional Divers, Ltd."
  msgInfo("Success", "Corrected spelling error.")
else
  msgInfo("Search Failed",
          "Couldn't find \nProfessional Divers, Ltd.")
endIf
ordTC.endEdit()
endMethod
```

locateNext method

Searches for a specified field value.

Syntax

```
1. locateNext ( const fieldName String, const exactMatch AnyType [ , const fieldName String,
const exactMatch AnyType ] * ) Logical

2. locateNext ( const fieldNum SmallInt, const exactMatch AnyType [ , const fieldNum SmallInt,
const exactMatch AnyType ] * ) Logical
```

Description

locateNext searches a table for values that match the criteria specified in one or more field value pairs. Specify the value to search for in *exactMatch* and the field to search in *fieldName* or *fieldNum*. This method guarantees that the first value matching *exactMatch* is found and given the current view of the records. If the TCursor is using a secondary index, **locate** finds the first record in the secondary index order.

The search begins at the top of the table, but if no match is found, the TCursor returns to the original record. If a match is found, the TCursor moves to that record. This operation fails if the active record cannot be posted (e.g., because of a key violation).

Note

- The search is case-sensitive unless **ignoreCaseInLocate** (Session type) is set to True.

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCLOCA;OPAL_METH_TCLNPA;OPAL_METH_TCLPAT;',0,"Defaultoverview",)} Related Topics

locateNext example

The following example uses **locate** and **locateNext** to count the number of records that have FL in the State/Prov field of the *Customer* table. The following code is attached to the **pushButton** method for *findFL* :

```
; findFL::pushButton
method pushButton(var eventInfo Event)
var
  CustTC TCursor
  numFound LongInt
endVar
custTC.open("Customer.db")


if custTC.locate("State/Prov", "FL") then
  numFound = 1
  while custTC.locateNext("State/Prov", "FL")
    numFound = numFound + 1
  endWhile
  msgInfo("Records Found", String("Found ", numFound, " companies in FL"))
else
  msgInfo("Sorry", "Can't find FL in State/Prov field.")
endIf


endMethod
```

locateNextPattern method

Locates the next record containing a field that has a specified pattern of characters.

Syntax

```
1. locateNextPattern ( [ const fieldName String, const exactMatch AnyType ] * const fieldName String, const pattern AnyType ) Logical

2. locateNextPattern ( [ const fieldNum SmallInt, const exactMatch AnyType ] * const fieldNum SmallInt, const pattern AnyType ) Logical
```

Description

locateNextPattern finds sub-strings (e.g., comp in computer). The search begins with the record after the active record. If a match is found, the TCursor moves to that record. If no match is found, the TCursor returns to the original record. If the TCursor is using a secondary index, **locateNextPattern** finds the next record in secondary index order regardless of that record's primary index order.

This operation fails if the active record cannot be committed (e.g., because of a key violation). To start a search at the beginning of a table, use **locatePattern**.

To search for records by the value of a single field, specify the field in *fieldName* or *fieldNum* (use *fieldNum* for faster performance) and specify a pattern of characters in *pattern*.

You can include the standard pattern operators **@** and **..** in the *pattern* argument. The **..** operator specifies any string of characters (including no string). The **@** operator specifies for any single character. Any combination of literal characters and wildcards can be used to construct a search. If **advancedWildCardsInLocate** (Session type) is enabled, you can use advanced match pattern operators. For more information, see the description of **advMatch**.

For example, the following statement examines the values in the first field of each record. If a value is anything except Corel, **locateNextPattern** returns True.

```
tc.locateNextPattern(1, [^Corel])
```

To search for records by the values of more than one field, specify exact matches on all fields except the last one in the list. For example, the following code searches the Name field for exact matches on the word Corel, the Product field for Corel Paradox, and the Keywords field for words beginning with data (e.g., database).

```
tc.locateNextPattern("Name", "Corel"  "Product", "Corel Paradox" "Keywords", "data..")
```

For examples, see Sample search strings with wildcards in the User's Guide help.

Note

- The search is case-sensitive unless **ignoreCaseInLocate** (Session type) is set to True.

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCLPAT;OPAL_METH_TCLNEX;OPAL_METH_STAMAT;OPAL_METH_SSADVANCEDWILDCARDSINLOCATE;OPAL_METH_SSISADVANCEDWILDCARDSINLOCATE;',0,"Defaultoverview",)} **Related Topics**

locateNextPattern example

In the following example, assume the SOFTWARE.DB table exists in the current directory. Assume further that two of the fields are named Product and Name. This code searches for records whose Name field contains Corel and whose Product field begins with Par. This code keeps track of the matches found and stores field values in a resizeable array. If the method can't locate any more records that match the criteria, the results are displayed in a dialog box. The following code is attached to a button's **pushButton** method:

```
; findGoodProducts::pushButton
method pushButton(var eventInfo Event)
var
  myNames TCursor
  searchFor String
  numFound SmallInt
  productNames Array[] String
endVar
myNames.open("software.db")
searchFor = "Corel"


; this searches for records with "Corel" in the Name field
; and values starting with "Par" in the Product field
if myNames.locatePattern("Name", searchFor, "Product", "Par..") then
  numFound = 1
  productNames.grow(1)
  productNames[numFound] = myNames.Product


  ; now continue searching through fields with same criteria and
  ; store Product values in productNames array
  while myNames.locateNextPattern("Name", searchFor, "Product", "Par..")
    numFound = numFound + 1
    productNames.addLast(myNames.product)
  endWhile
endIf
if productNames.size() > 0 then
  productNames.view()
endIf
endMethod
```

locatePattern method

Locates a record containing a field that has a specified pattern of characters.

Syntax

```
1. locatePattern ( [ const fieldName String, const exactMatch AnyType ] * const fieldName String, const pattern String ) Logical

2. locatePattern ( [ const fieldNum SmallInt, const exactMatch AnyType ] * const fieldNum SmallInt, const pattern String ) Logical
```

Description

locatePattern finds sub-strings (e.g., comp in computer). The search always starts at the beginning of the table, but if no match is found, the TCursor returns original record. If a match is found, the TCursor moves to that record. If the TCursor is using a secondary index, locate finds the first record in secondary index order regardless of that record's primary index order.

This operation fails if the active record cannot be committed (e.g., because of a key violation). To start a search after the active record, use **locateNextPattern**. To start a search before the active record, use **locatePriorPattern**.

To search for records by the value of a single field, specify the field in *fieldName* or *fieldNum* (use *fieldNum* for faster performance) and specify a pattern of characters in *pattern*.

You can include the standard pattern operators **@** and **..** in the *pattern* argument. The **..** operator specifies any string of characters (including no string). The **@** operator specifies for any single character. Any combination of literal characters and wildcards can be used to construct a search. If **advancedWildCardsInLocate** (Session type) is enabled, you can use advanced match pattern operators. For more information, see the description of **advMatch**.

For example, the following statement examines values in the first field of each record. If a value is anything except Corel, **locatePattern** returns True.

```
tc.locatePattern(1, [^Corel])
```

To search for records by the values of more than one field, specify exact matches on all fields except the last one in the list. For example, the following code searches the Name field for exact matches on the word Corel, the Product field for Corel Paradox, and the Keywords field for words beginning with data (e.g., database).

To start a search from the beginning of a table, use **locateNextPattern**:

```
tc.locateNextPattern("Name", "Corel"  "Product", "Corel Paradox" "Keywords", "data..")
```

For examples, see Sample search strings with wildcards in the User's Guide help.

Note

- The search is case-sensitive unless **ignoreCaseInLocate** (Session type) is set to True.

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCLNPA;OPAL_METH_TCLOCA;OPAL_METH_STAMAT;OPAL_METH_SSADVANCEDWILDCARDSINLOCATE;OPAL_METH_SSISADVANCEDWILDCARDSINLOCATE;',0,"Defaultoverview",)} Related Topics

locatePattern example

In the following example, assume the SOFTWARE.DB table exists in the current directory. Assume further that two of the fields are named Product and Name. This code searches for records whose Name field contains Corel and whose Product field begins with Par. This code keeps track of the matches found and stores field values in a resizeable array. If the method can't locate any more records that match the criteria, the results are displayed in a dialog box. The following code is attached to a button's **pushButton** method:

```
; findGoodProducts::pushButton
method pushButton(var eventInfo Event)
var
  myNames TCursor
  searchFor String
  numFound SmallInt
  productNames Array[] String
endVar
myNames.open("software.db")
searchFor = "Corel"


; this searches for records with "Corel" in the Name field
; and values starting with "Par" in the Product field
if myNames.locatePattern("Name", searchFor, "Product", "Par..") then
  numFound = 1
  productNames.grow(1)
  productNames[numFound] = myNames.Product


  ; now continue searching through fields with same criteria and
  ; store Product values in productNames array
  while myNames.locateNextPattern("Name", searchFor, "Product", "Par..")
    numFound = numFound + 1
    productNames.addLast(myNames.product)
  endWhile
endIf
if productNames.size() > 0 then
  productNames.view()
endIf
endMethod
```

locatePrior method

Searches for a specified field value.

Syntax

```
1. locatePrior ( const fieldName String, const exactMatch AnyType
                 [ , const fieldName String, const exactMatch AnyType ] * ) Logical

2. locatePrior ( const fieldNum SmallInt, const exactMatch AnyType
                 [ , const fieldNum SmallInt, const exactMatch AnyType ] * ) Logical
```

Description

locatePrior searches backwards from the active record in a table for record values that match one or more field/value pairs. Specify the search value in *exactMatch* and the search field in *fieldName* or *fieldNum* (use *fieldNum* for faster performance). This method guarantees that the previous value matching *exactMatch* is found, given the current view of the records. If the TCursor is using a secondary index, **locatePrior** finds the previous record in secondary index order.

The search begins with the record before the active record and moves up through the table. If a match is found, the TCursor moves to that record. This operation fails if the active record cannot be posted and unlocked (e.g., due to a key violation). If no match is found, the cursor returns to the active record. This method returns True if a successful match was made; otherwise, it returns False.

Note

- The search is case-sensitive unless **ignoreCaseInLocate** (Session type) is set to True.

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCLNEX;OPAL_METH_TCLNPA;OPAL_METH_TCLPAT;',0,"Defaultoverview",)} Related Topics

locatePrior example

In the following example, the **pushButton** method for *showPrior* searches backwards through the *Lineitem* table for records with a certain order number. The *lineTC* variable is declared in the page's Var window, and opened to the *Lineitem* table in the **open** method for the page.

The following code goes in the Var window for *thisPage*:

```
; thisPage::var
Var
  lineTC TCursor
endVar
```

The following code is attached to the **open** method for *thisPage*:

```
; thisPage::open
method open(var eventInfo Event)
  lineTC.open("Lineitem")        ; open a TCursor for LineItem.db
endMethod
```

The following code is attached to the **pushButton** method for the *showPrior* button:

```
; showPrior::pushButton
method pushButton(var eventInfo Event)
var
  rec Array[] AnyType
endVar


if lineTC.locatePrior("Order No", 1005) then
  lineTC.copyToArray(rec)
  rec.view("Record #" + String(lineTC.recNo()))
else
  msgStop("Sorry", "No more records.")
endIf
endMethod
```

locatePriorPattern method

Locates the previous record containing a field that has a specified pattern of characters.

Syntax

```
1. locatePriorPattern ( [ const fieldName String, const exactMatch AnyType ] * const fieldName String, const pattern String ) Logical

2. locatePriorPattern ( [ const fieldNum SmallInt, const exactMatch AnyType ] * const fieldNum SmallInt, const pattern String ) Logical
```

Description

locatePriorPattern finds sub-strings (e.g., comp in computer). The search begins with the record before the active record. If a match is found, the TCursor moves to that record. If no match is found, the TCursor returns to the original record. If the TCursor is using a secondary index, **locatePriorPattern** finds the previous record in secondary index order regardless of that record's primary index order.

This operation fails if the active record cannot be committed (e.g., due to a key violation). If no match is found, the cursor returns to the active record. To start a search at the beginning of a table, use **locatePattern**.

To search for records by the value of a single field, specify the field in *fieldName* or *fieldNum* (use *fieldNum* for faster performance) and specify a pattern of characters in *pattern*.

You can include the standard pattern operators **@** and **..** in the *pattern* argument. The **..** operator specifies any string of characters (including no string). The **@** operator specifies for any single character. Any combination of literal characters and wildcards can be used to construct a search. If **advancedWildCardsInLocate** (Session type) is enabled, you can use advanced match pattern operators. For more information, see the description of **advMatch**.

For example, the following statement examines values in first field of each record. If a value is anything except Corel, locatePriorPattern returns True.

```
tc.locatePriorPattern(1, [^Corel])
```

To search for records by the values of more than one field, specify exact matches on all fields except the last one in the list. For example, the following code searches the Name field for exact matches on the word Corel, the Product field for Corel Paradox, and the Keywords field for words beginning with data (e.g., database).

To start a search from the beginning of a table, use **locateNextPattern**.

```
tc.locateNextPattern("Name", "Corel"  "Product", "Corel Paradox" "Keywords", "data..")
```

For examples, see Sample search strings with wildcards in the User's Guide help.

Note

- The search is case-sensitive unless **ignoreCaseInLocate** (Session type) is set to True.

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCLPAT;OPAL_METH_TCLNPA;OPAL_METH_STAMAT;OPAL_METH_SSADVANCEDWILDCARDSINLOCATE;OPAL_METH_SSISADVANCEDWILDCARDSINLOCATE;',0,"Defaultoverview",)} Related Topics

locatePriorPattern example

In the following example, the **pushButton** method for *showPriorPtrn* searches backwards through the *Software* table for records with a certain company and product name. The *tc* variable is declared in the page's Var window, and opened to the *Software* table in the **open** method for the page.

The following code goes in the Var window for *thisPage*:

```
; thisPage::var
Var
  tc        TCursor
  searchFor String
endVar
```

The following code is attached to the **open** method for *thisPage*:

```
; thisPage::open
method open(var eventInfo Event)
  tc.open("Software.db")  ; open TCursor for Software.db
  tc.end()                ; move TCursor to the last record
  searchFor = "Corel"
endMethod
```

The following code is attached to the **pushButton** method for the *showPriorPtrn* button:

```
; showPrior::pushButton
method pushButton(var eventInfo Event)
var
  rec Array[] AnyType
endVar


; search for the previous pattern
if tc.locatePriorPattern("Name", searchFor, "Product", "Par..") then
  tc.copyToArray(rec)
  rec.view("Record #" + String(tc.recNo()))
else
  msgStop("Sorry", "No more records.")
endIf
endMethod
```

lock method

Places specified locks on a table.

Syntax

```
lock ( const lockType String ) Logical
```

Description

lock places locks on the TCursor. The *lockType* argument is one of the following String values, listed in order of decreasing strength and increasing concurrency.

String value	Description
Full	The current session has exclusive access to the table. Cannot be used with dBASE tables.
Write	The current session can write to and read from the table. No other session can place a write lock or a read lock on the table.
Read	The current session can read from the table. No other session can place a write lock, full lock, or exclusive lock on the table.

If successful, **lock** returns True; otherwise, it returns False.

Example

**{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCLSTA;OPAL_METH_TCUNLOCK;',0,"Defaultoverview",)
} Related Topics**

lock example

The following example attaches a Table variable to *Customer*, places an exclusive lock on the table and uses **reIndex** to rebuild the *Phone_Zip* index. When the index is rebuilt, this code unlocks *Customer* so other network users can gain access to the table.

```
; reindexCust::pushButton
method pushButton(var eventInfo Event)
var
  tc TCursor
  pdoxTbl String
endVar
pdoxTbl = "Customer.db"


if tc.open(pdoxTbl) then
  if tc.lock("Full") then     ; attempt to place Full lock
    tc.reIndex("Phone_Zip")   ; rebuild Phone_Zip index
    tc.unLock("Full")         ; unlock the table
    message("Phone_Zip rebuilt.")
  else
    msgStop("Sorry", "Can't lock " + pdoxTbl + " table.")
  endIf
endIf
endMethod
```

lockRecord method

Puts a write lock on the active record.

Syntax

lockRecord ( ) Logical

Description

lockRecord attempts to place a write lock on the record pointed to by a TCursor (an explicit record lock). **lockRecord** returns True if successful; otherwise, it returns False.

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCUNREC;',0,"Defaultoverview",)} **Related Topics**

lockRecord example

In the following example, the **pushButton** method for *thisButton* searches for a record in the *Customer* table. If the search is successful, this code locks the record using **lockRecord**. When the record has been locked, a custom procedure is called to get new customer information from the user. If **lockRecord** is not successful, the user is asked to try again later.

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
  custTC, myCustTC TCursor
endVar
custTC.open("Customer.db")


; attempt to locate record in Customer.db
if custTC.locatePattern("Name", "Jamaica..") then
  custTC.edit()
  if custTC.lockRecord() then        ; attempt to lock the record
    custTC.initRecord()              ; initialize record to the
                                     ; defaults
    getCustInfo()                    ; call a custom procedure
  else                               ; otherwise record couldn't be
                                     ; locked
    msgStop("Sorry", "Can't lock record. \n Try again later.")
  endIf
else
  msgStop("Sorry", "Can't find record.")
endIf

endMethod
```

lockStatus method

Returns the number of locks on a TCursor.

Syntax

```
lockStatus ( lockType String ) SmallInt
```

Description

lockStatus returns the number of times you have placed a lock of type *lockType* on a TCursor. *lockType*'s value is Write, Read, or Any.

If you haven't placed any locks on the table **lockStatus** returns 0.

If you specify Any for *lockType*, **lockStatus** returns the total number of locks you've placed on the TCursor. **lockStatus** does not include locks placed by Corel Paradox or by other users or applications.

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCLOCK;OPAL_METH_TCUNREC;',0,"Defaultoverview",)}

Related Topics

lockStatus example

The following example uses **lockStatus** to report on locks you've placed explicitly on a TCursor. Assume a form contains a button named *thisButton* and a field object named *Balance_Due* that is bound to the Balance Due field of the *Orders* table.

```
; thisButton::pushButton
const
   kTbName = "locks"
   kStatus = "Any"
endConst


var
   tcOrders   TCursor
   tvLocks   TableView
endVar


proc displayLockInfo()
   tcOrders.enumLocks(kTbName)
   tvLocks.open(kTbName)
   tvLocks.setTitle("Locks on Orders table:")


   siNumLocks = tcOrders.lockStatus(kStatus)
   siNumLocks.view("Locks on TCursor:")
   tvLocks.close()
endProc


method pushButton(var eventInfo Event)


   ; Associate TCursor with a field object bound
   ; to the Balance Due field in the Orders table.
   ; TCursor gets locks from the UIObject.
   tcOrders.attach(Balance_Due)

   displayLockInfo() ; Table is locked, but not TCursor.


   tcOrders.lock("Write") ; Lock TCursor.


   displayLockInfo() ; Table and TCursor are locked,
                     ; but locks are different.
endmethod
```

moveToRecNo method

Moves a TCursor to a specific record.

Syntax

```
moveToRecNo ( const recordNum LongInt ) Logical
```

Description

moveToRecNo moves to the record specified in *recordNum*. This method returns an error if *recordNum* doesn't exist. Use the **nRecords** method or examine the NRecords property to determine the number of records in a table. This method is recommended only for dBASE tables. If used for a Corel Paradox table, **moveToRecNo** behaves exactly like the **moveToRecord** method.

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCCURE;OPAL_METH_TCEND;OPAL_METH_TCHOME;OPAL_METH_TCMTRE;OPAL_METH_TCNERE;OPAL_METH_TCNREC;OPAL_METH_TCPRRE;OPAL_METH_TCSKIP;',0,"Defaultoverview",)} Related Topics

moveToRecNo example

The following example uses **moveToRecNo** to move to a specified record in the dBASE table ORDERS.DBF. This code then displays the value of the SALE_DATE field for that record.

```
method pushButton(var eventInfo Event)
   var
      tcOrders   TCursor
      siRecNo      SmallInt
      daSaleDate   Date
   endVar


   tcOrders.open("orders.dbf")

   siRecNo = 0
   siRecNo.view("Enter a record number:")


   if siRecNo > 0 then
      if   tcOrders.moveToRecNo(siRecNo) then
         daSaleDate = tcOrders."SALE_DATE"
         daSaleDate.view("Sale date: ")
      else
         errorShow("Invalid record number.")
      endIf
   else
      return
   endIf


endMethod
```

moveToRecord method

Moves a TCursor to a specific record in a table.

Syntax

```
moveToRecord ( const recordNum LongInt ) Logical
```

Description

moveToRecord moves a TCursor to the record specified in *recordNum*. This method returns an error if *recordNum* is greater than the number of records in the table. Use **nRecords** to determine how many records a table contains. This method can be very slow for dBASE tables; use **moveToRecNo** instead.

This operation fails if the active record cannot be committed (e.g., because of a key violation).

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCNREC;OPAL_METH_TCHOME;OPAL_METH_TCEND;OPAL_METH_TCNERE;OPAL_METH_TCPRRE;OPAL_METH_TCCURE;OPAL_METH_TCSKIP;',0,"Defaultoverview",)}

Related Topics

moveToRecord example

The following example uses **moveToRecord** to move to a specified record in the *Orders* table. This code then displays the value of the Sale Date field for the specified record:

```
method pushButton(var eventInfo Event)
   var
      tcOrders   TCursor
      siRecNo      SmallInt
      daSaleDate   Date
   endVar


   tcOrders.open("orders.db")

   siRecNo = 0
   siRecNo.view("Enter a record number:")


   if siRecNo > 0 then
      if   tcOrders.moveToRecord(siRecNo) then
         daSaleDate = tcOrders."Sale Date"
         daSaleDate.view("Sale date: ")
      else
         errorShow("Invalid record number.")
      endIf
   else
      return
   endIf


endMethod
```

nextRecord method

Moves to the next record in a table.

Syntax

nextRecord ( ) Logical

Description

nextRecord moves the TCursor to the next record in the table. If the table is in Edit mode, **nextRecord** commits changes to the active record before moving. This operation fails if the active record cannot be committed (e.g., because of a key violation).

If you attempt to move past the end of the table, **nextRecord** returns False, the last record of the table becomes the active record, and **eot** returns True.

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCHOME;OPAL_METH_TCEND;OPAL_METH_TCPRRE;OPAL_METH_TCSKIP;OPAL_METH_TCMTRE;',0,"Defaultoverview",)} **Related Topics**

nextRecord example

In the following example, the **pushButton** method for *showNextCust* uses **nextRecord** to move a TCursor through the *Customer* table. Each time the TCursor lands on a new record, the code uses **copyToArray** to copy the contents of the record to a dynamic array (DynArray) and displays field values in a dialog box. When **nextRecord** attempts to move past the last record in the table, **eot** returns True and the **pushButton** method terminates.

```
; showNextCust::pushButton
method pushButton(var eventInfo Event)
var
  tc TCursor
  scratch DynArray[] AnyType
  tblName String
endVar
tblName = "Customer.db"


if tc.open(tblName) then


  while NOT tc.eot()           ; True until nextRecord attempts to move
                               ; beyond the end the table
    tc.copyToArray(scratch)    ; copy the record to scratch DynArray
    scratch.view("Record " + String(tc.recNo()))
    if msgQuestion("",
       "Do you want to see the next record?") = "Yes" then
      tc.nextRecord()          ; move down one record
    else
      return
    endIf
  endWhile


  msgStop("That's it!", "No more records.")


else
  msgStop("Sorry", "Can't open " + tblName + " table.")
endIf
endMethod
```

nFields method

Returns the number of fields in a table.

Syntax

```
nFields ( ) LongInt
```

Description

nFields returns the number of fields in the table associated with a TCursor.

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCNKFI;OPAL_METH_TCNREC;',0,"Defaultoverview",)}

Related Topics

nFields example

In the following example, the **pushButton** method for *thisButton* displays the number of fields in the *BioLife* table:

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
  tc TCursor
endVar
if tc.open("BioLife.db") then
  msgInfo("Number of BioLife fields", tc.nFields())
else
  msgStop("Sorry", "Can't open BioLife.db table")
endIf

endMethod
```

nKeyFields method

Returns the number of fields in the index of a table.

Syntax

```
nKeyFields ( ) LongInt
```

Description

nKeyFields returns the number of fields in the active index of the table associated with a TCursor. Use **getIndexName** to get the name of the current index.

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCGETINDEXNAME;OPAL_METH_TCNFLD;OPAL_METH_TCNREC;',0,"Defaultoverview",)} Related Topics

nKeyFields example

The following example reports the number of key fields in a Corel Paradox table:

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
  pdoxTC   TCursor
  nkf      LongInt
  pdoxTbl  String
endVar
pdoxTbl = "Orders.db"


if pdoxTC.open(pdoxTbl) then
  nkf = pdoxTC.nKeyFields() ; Key fields in the primary index
  msgInfo(pdoxTbl,
          pdoxTbl + " has " + String(nkf) + " key fields.")
else
  msgInfo("Sorry", "Can't open " + pdoxTbl + " table.")
endIf


endMethod
```

nRecords method

Returns the number of records in a table.

Syntax

```
nRecords ( ) LongInt
```

Description

nRecords returns the number of records in the table associated with a TCursor. This operation can take a long time for dBASE tables and large Corel Paradox tables.

If working with a dBASE table, **nRecords** counts deleted records if **showDeleted** is turned on. Otherwise, deleted records are not counted.

Notes

- When you call **nRecords** after setting a filter, the returned value does *not* represent the number of records in the filtered set. To get that information, use **cCount**.
- When you call **nRecords** after setting a range, the returned value represents the number of records in the set defined by the range.

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCNFLD;OPAL_METH_TCNKFI;',0,"Defaultoverview",)}

Related Topics

nRecords example

In the following example, the **pushButton** method for *thisButton* runs a custom method. If there are more than 10,000 records in ORDERS.DB; otherwise, this code displays the current number of records in *Orders*.

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
  ordTC TCursor
  nOrders LongInt
endVar
if ordTC.open("Orders.db") then
  nOrders = ordTC.nRecords()
  if nOrders > 10000 then   ; If Orders has more than 10,000 records
    archiveOldOrders()      ; run a custom method.
  else
    msgInfo("Status",
            "Orders table has " + String(nOrders) + " records.")
  endIf
else
  msgStop("Sorry", "Can't open Orders table.")
endIf
endMethod
```

open method

Opens a TCursor on a table.

Syntax

```
1. open ( const tableName String [ , const db DataBase ] [ , const indexName String ] ) Logical
2. open ( const tableVar Table ) Logical
```

Description

open associates a TCursor with the table named in *tableName*.

In Syntax 1, *tableName* is a String and you can use arguments *db* and *indexName* to specify a database and an index. If *tableName* does not specify a filename extension, Corel Paradox assumes the extension is .DB.

In Syntax 2, *tableVar* is the name of a Table variable. You can use the Table method **setIndex** to specify an index, and you can specify the database using the Table method **attach**.

Examples

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCCLOS;',0,"Defaultoverview",)} Related Topics

open examples

Example1	Using the first open syntax
Example2	Using the second open syntax

open example 1

The following example uses the Syntax 1 to open a TCursor on the *Customer* table in the SampleTables database. This code uses the optional *indexName* clause, so the TCursor uses the NameAndState index. The following code is attached to the **pushButton** method for *firstButton*:

```
; firstButton::pushButton
method pushButton(var eventInfo Event)
var
  tc1  TCursor
  samp Database
endVar


; Create the SampleTables alias for the default sample directory.
addAlias("SampleTables", "Standard", "c:\\Corel\\Suite8\\Paradox\\samples")


; Associate the samp Database var with SampleTables database.
samp.open("SampleTables")


; Associate tc1 to the Customer table in samp database,
; and use the NameAndState index.
if not tc1.open("Customer.db", samp, "NameAndState") then
   errorShow()
endIf

endMethod
```

open example 2

The following example uses Syntax 2 to open a TCursor. The following code is attached to the **pushButton** method for *secondButton*:

```
; secondButton::pushButton
method pushButton(var eventInfo Event)
var
  tc1  TCursor
  samp DataBase
  tbl  Table
endVar


; Create the SampleTables alias for the default sample directory.
addAlias("SampleTables", "Standard", "c:\\Corel\\Suite8\\Paradox\\samples")


; Associate the samp DataBase var with SampleTables database.
samp.open("SampleTables")


; Attach the tbl Table handle to Customer in the samp database.
tbl.attach("Customer.db", samp)
; Set the tbl index to the NameAndState index.
tbl.setIndex("NameAndState")


; Now associate tc1 TCursor to Customer table in samp database.
if not tc1.open(tbl) then
   errorShow()
endIf


endMethod
```

postRecord method

Posts changes to a record.

Syntax

postRecord ( ) Logical

Description

postRecord posts changes to a record immediately. The record remains locked throughout the posting process. If a key value changes in an indexed table and the record flies away, the corresponding TCursor flies with it. This method returns True if successful; otherwise, it returns False.

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCUNREC;',0,"Defaultoverview",)} **Related Topics**

postRecord example

In the following example, the **pushButton** method for the *fixName* button attempts to find a misspelled name in the *Customer* table. If the erroneous name is found, the code corrects it and posts changes using **postRecord**.

```
; fixName::pushButton
method pushButton(var eventInfo Event)
var
  tc TCursor
  badName String
endVar
badName = "Usco"
goodName = "Unisco"


tc.open("Customer.db")
if tc.locate("Name", badName) then ; if the erroneous name is found
  tc.edit()                        ; put TCursor in Edit mode
  tc.Name = goodName               ; correct misspelled name
  if tc.postRecord() then          ; True if record is posted
    message("Changes posted.")
  else                             ; record is not posted (Key violation?)
    msgStop("PostRecord", "Can't post these changes.")
  endIf
  tc.endEdit()                     ; end Edit mode
  ; If the record was committed, endEdit simply ends Edit mode. The Name
  ; field now stores "Unisco". If the record was not committed, the field
  ; retains its original value ("Usco").


else                               ; can't find "Usco" in Name field
   message("Can't find " + badName)
endIf
endMethod
```

priorRecord method

Moves to the previous record in a table.

Syntax

```
priorRecord ( ) Logical
```

Description

priorRecord sets the active record to the previous record in a table. If the table is in Edit mode, **priorRecord** commits changes to the active record before moving. This method returns False if the TCursor is already at the first record. Also, the first record of the table becomes the active record, and **bot** returns True.

priorRecord may not be appropriate in all databases, because some may not be bi-directional. This operation fails if the active record cannot be committed (e.g., because of a key violation).

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCHOME;OPAL_METH_TCEND;OPAL_METH_TCNERE;OPAL_METH_TCSKIP;OPAL_METH_TCMTRE;',0,"Defaultoverview",)} Related Topics

priorRecord example

In the following example, the **pushButton** method for *showPrevCust* uses **priorRecord** to move a TCursor back through the *Customer* table. Each time the TCursor lands on a new record, this code uses **copyToArray** to copy the record's contents to a dynamic array (DynArray) and display field values in a dialog box. When **priorRecord** attempts to move beyond the beginning of the table, **bot** returns True and the **pushButton** method terminates.

```
; showPrevCust::pushButton
method pushButton(var eventInfo Event)
var
  tc TCursor
  scratch DynArray[] AnyType
  tblName String
endVar
tblName = "Customer.db"


if tc.open(tblName) then


  tc.end()                      ; move to end of table
  while NOT tc.bot()            ; True until priorRecord attempts to move
                                ; beyond the beginning of the table
    tc.copyToArray(scratch)     ; copy the record to scratch DynArray
    scratch.view("Record " + String(tc.recNo()))
    if msgQuestion("",
       "Do you want to see the next record?") = "Yes" then
      tc.priorRecord()          ; move up one record
    else
      return
    endIf
  endWhile


  msgStop("That's it!", "No more records.")


else
  msgStop("Sorry", "Can't open " + tblName + " table.")
endIf
endMethod
```

qLocate method

Searches an indexed table for a specified field value.

Syntax

```
qLocate ( const searchValue AnyType [ , const searchValue AnyType ] * ) Logical
```

Description

qLocate searches an indexed table for records which have key field values that exactly match the criteria specified in *searchValue*. **qLocate** searches for values in the active index (the first value corresponds to the index's first field, the second value corresponds to the index's second field, and so on).

The search always starts from the beginning of the table. If no match is found, the TCursor position is set to where it would be if there had been a match. If a match is found, the TCursor moves to that record. This method does not attempt to post the active record. The operation fails if the number of search values exceeds the number of fields in the current index.

qLocate does not clear existing record locks on the TCursor. If a lock is present, **qLocate** will fail. To prevent failure, issue an **unLockRecord** before the **qLocate** is called. This could be particularly helpful within a scan loop.

Note

- **qlocate** can be used to simulate incremental searches. If **qlocate** finds a matching record for *searchValue*, the TCursor position is set to that record. If **qlocate** fails to find a match, the TCursor position is left where it would have been had there been a match.

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCLOCA;OPAL_METH_TCLNEX;OPAL_METH_TCLNPA;OPAL_METH_TCLPAT;OPAL_METH_TCLOCATEPRIOR;OPAL_METH_TCLOCATEPRIORPATTERN;',0,"Defaultoverview",)} **Related Topics**

qLocate example

The following example uses **qLocate** to find a key value in the *Lineitem* table:

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
  tc TCursor
endvar


if tc.open("Lineitem.db") then


  ; if qLocate can find 1002 in the first field of the
  ; index and 1316 in the second field of the index
  if tc.qLocate(1002, 1316) then


    ; make some changes to the record
    tc.edit()
    tc.Qty = 10
    tc.Total = tc."Selling Price" * tc.Qty
    tc.close()
  else
    msgStop("Sorry", "Can't find specified record.")
  endIf
else
  msgStop("Error", "Can't open Lineitem.db")
endIf


endMethod
```

recNo method

Returns the record number of the active record.

Syntax

```
recNo ( ) LongInt
```

Description

recNo returns an integer representing the active record's position in the table. For a dBASE table, **recNo** returns the physical position of the record in the table; for an indexed Corel Paradox table, it returns the record's sorted position according to the current index.

Note

- When you call **recNo** after setting a filter, the returned value is represented by the ObjectPAL constant peInvalidRecordNumber.

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCNREC;',0,"Defaultoverview",)} Related Topics

recNo example

In the following example, the **pushButton** method for *thisButton* searches the *Customer* table for customers that reside in Oregon. If Oregon residents are found, this code stores record numbers in an array and displays the array in a dialog box:

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
  tc TCursor
  ar Array[] SmallInt
  tblName String
endVar
tblName = "Customer.db"


tc.open(tblName)
if tc.locate("State/Prov", "OR") then
  ar.addLast(tc.recNo())                    ; add record number to array
  while tc.locateNext("State/Prov", "OR")   ; find the next "OR"
    ar.addLast(tc.recNo())                  ; add more array elements
  endWhile
  ar.view("Record Numbers")                 ; display ar array
else
  msgInfo("Nothing to do!", "Can't find \"OR\" in \"State/Prov\" field")
endIf
endMethod
```

recordStatus method

Reports the status of a record.

Syntax

```
recordStatus ( const statusType String ) Logical
```

Description

recordStatus returns True or False answers to a question about the status of a record. Use the argument *statusType* to specify the status in question (i.e., is New, Locked, or Modified).

The New value means the record has just been added to the table. Locked means that an implicit or explicit lock has been placed on the record. Modified means at least one of the field values has been changed and is not yet posted to the table.

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCLREC;OPAL_METH_TCUNREC;',0,"Defaultoverview",)}

Related Topics

recordStatus example

The following example determines whether the active record is locked. If the record is not locked, this code uses **lockRecord** to lock the record; otherwise this code informs the user:

```
; lockThisRecord::pushButton
method pushButton(var eventInfo Event)
var
  tc TCursor
endVar
tc.open("orders.db")
tc.edit()


; if the active record is NOT locked
if tc.recordStatus("Locked") = False then
  ; lock the active record
  tc.lockRecord()


  ; if record is locked, this statement will display True
  msgInfo("Record Status", "recordStatus(\"Locked\") = " +
                        String(tc.recordStatus("Locked")))
else
  message("Active record is already locked.")
endIf


endMethod
```

reIndex method

Rebuilds an index or index tag that is not automatically maintained.

Syntax

```
reIndex ( const IndexName String [ , const TagName String ] ) Logical
```

Description

reIndex rebuilds an index or index tag that is not automatically maintained. In a Corel Paradox table, use *indexName* to specify an index. In a dBASE table, use *indexName* to specify an .NDX file, or *indexName* and *tagName* to specify an index tag in an .MDX file. **reIndex** requires exclusive access to the table.

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCREINAL;',0,"Defaultoverview",)} **Related Topics**

reIndex example

The following example opens a TCursor for *Customer* (a Corel Paradox table), gains exclusive access to the table and uses **reIndex** to rebuild the *Phone_Zip* index:

```
; reindexCust::pushButton
method pushButton(var eventInfo Event)
var
  tc      TCursor
  pdoxTbl String
  tb      Table
endVar
pdoxTbl = "Customer.db"


tb.attach(pdoxTbl)
tb.setExclusive(Yes)


if tc.open(tb) then
  tc.reIndex("Phone_Zip")          ; rebuild Phone_Zip index
  message("Phone_Zip reindexed.")
else
  msgStop("Sorry", "Can't open " + pdoxTbl + " table.")
endIf


endMethod
```

reIndexAll method

Rebuilds all index files for a table.

Syntax

reIndexAll ( ) Logical

Description

reIndexAll rebuilds all indexes for the table associated with a TCursor. This method requires exclusive rights to rebuild a maintained index and a write lock to rebuild a non-maintained index. **reIndexAll** works only with Corel Paradox tables, because any index opened for a dBASE table is maintained automatically.

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCREIND;',0,"Defaultoverview",)} Related Topics

reIndexAll example

The following example rebuilds all indexes for the *Customer* table:

```
; reindexAllCust::pushButton
method pushButton(var eventInfo Event)
var
  tc      TCursor
  pdoxTbl String
  tb      Table
endVar
pdoxTbl = "Customer.db"


tb.attach(pdoxTbl)
tb.setExclusive(Yes) ; Need exclusive rights for a  maintained index.


if tc.open(tb) then
  tc.reIndexAll()               ; Rebuild all Customer indexes.
  message("Indexes rebuilt.")
else
  msgStop("Sorry", "Can't open " + pdoxTbl + " table.")
endIf
endMethod
```

seqNo method

Returns the record number of the active record.

Syntax

```
seqNo ( ) LongInt
```

Description

seqNo returns an integer representing the active record's position in a table. For dBASE tables, **seqNo** returns the sequential position of a record as viewed by the current index. **seqNo** and **recNo** always return the same value for Corel Paradox tables.

Note

- If you call **seqNo** after setting a filter, the return value is represented by the ObjectPAL constant named peInvalidRecordNumber.

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCMTRENO;OPAL_METH_TCMTRE;OPAL_METH_TCRECNO;',0,"Defaultoverview",)} Related Topics

seqNo example

The following example assumes that SCORES.DBF has three records and that the second record has been deleted. This code attaches to the **pushButton** method for *testSeqNo* and demonstrates the difference between **seqNo** and **recNo** methods:

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
  tc TCursor
endVar


  ; Scores.dbf has 3 records and the second record is deleted
tc.open("Scores.dbf")


  ; do not show deleted records
tc.showDeleted(No)


  ; this displays recNo() = 1
  ;               seqNo() = 1
msgInfo("tc Status", "recNo() = " + String(tc.recNo()) + "\n" +
                     "seqNo() = " + String(tc.seqNo()))


  ; move to the last record in the table
tc.end()


  ; this displays   recNo() = 3
  ;                 seqNo() = 2   (record number 2 is deleted)
msgInfo("tc Status", "recNo() = " + String(tc.recNo()) + "\n" +
                     "seqNo() = " + String(tc.seqNo()))


endMethod
```

setBatchOff method

Ends the batch processing mode invoked by a call to **setBatchOn**.

Syntax

```
setBatchOff ( ) Logical
```

Description

setBatchOff ends the batch processing mode by removing the restrictions imposed by **setBatchOn**.

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCSETBATCHON;',0,"Defaultoverview",)} **Related Topics**

setBatchOff example

See the **setBatchOn** example.

setBatchOn method

Groups multiple operations to improve the performance of table updates in a multi-user environment.

Syntax

```
setBatchOn ( ) Logical
```

Description

setBatchOn groups multiple operations to improve the performance of table updates in a multi-user environment. If update operations are performed after executing a **setBatchOn** statement, file I/O and concurrency control are minimized, resulting in improved performance. **setBatchOn** grants you exclusive access to a table. After **setBatchOn** executes, no other user or session can access, open, modify, lock, or read from the table until **setBatchOff** executes. (Other TCursors in the same session can still access the table.) If **setBatchOff** does not execute, the lock remains in effect for the life of the TCursor. Use **setBatchOn** when several short operations should occur sequentially. **setBatchOn** should be used by advanced developers for serializing operations and improving performance. Most developers will not need this command.

Notes

- **setBatchOn** operates for less than two seconds. If another user attempts to update or access the current table, that user's system freezes. If **setBatchOn** is not followed by a **setBatchOff** statement, the other user's system remains frozen for up to two minutes. After two minutes, the operation that caused the user's system to freeze fails (due to a timeout error) and the user's system resumes operation.
- Other users cannot determine whether **setBatchOn** has been called. To minimize the chances of interfering with other users, call **setBatchOff** as soon as possible after calling **setBatchOn**.

Examples

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCSETBATCHOFF;',0,"Defaultoverview",)} Related Topics

setBatchOn method examples

Example1 Using **setBatchOn** to delete line items in a table

Example2 Serializing access to autosequence numbers

setBatchOn example 1

The following example assumes that a form's data model contains the *Orders* table and the *Lineitem* table linked 1:M, with *Orders* as the master table. This code deletes the records in the current detail set (the line items for the current order). In this example, *Lineitem* is a tableframe or a multi-record object that is bound to the Lineitem table:

```
method pushButton(var eventInfo Event)
   var
      ordersTC TCursor
   endVar


   ordersTC.attach(Lineitem) ; attach to the detail set
   ordersTC.edit()


   ordersTC.setBatchOn()
   while not ordersTC.eot()
      ordersTC.deleteRecord()
   endWhile
   ordersTC.setBatchOff()


endMethod
```

setBatchOn example 2

Many applications require an autosequence number that must be incremented by each user who attempts to add a record to a table. This code serializes access to an autosequence number using **setBatchOn** and **setBatchOff**. The following example assumes that the *NumTable* table contains a single numeric field named *Sequence Number*.

In this example, each user who attempts an operation calls the custom method **GetAutoSequence**. The first user who calls the method gets the lowest sequence number. The call to **setBatchOn** holds every other user out without locking the table. Every other user who has issued a GetAutoSequence call gains access to the table sequentially.

```
method GetAutoSequence() LongInt
   var
      numTableTC   TCursor
      SequenceVar   LongInt
   endVar


   numTableTC.open("numtable.db")
   numTableTC.edit()


   numTableTC.setBatchOn()
   numTableTC."Sequence Number" = numTableTC."Sequence Number" + 1
   numTableTC.postRecord()
   SequenceVar = numTableTC."Sequence Number"
   numTableTC.setBatchOff()


   return SequenceVar
endMethod
```

setFieldValue method

Assigns a value to a specified field.

Syntax

```
1. setFieldValue ( const fieldName String, const value AnyType ) Logical
2. setFieldValue ( const fieldNum SmallInt, const value AnyType ) Logical
```

Description

setFieldValue sets the value of the field specified by*fieldName* (or *fieldNum*) to *value*. This method returns True if successful; otherwise, it returns False.

You can also set the value of this field using dot notation. For example, this statement uses dot notation to change the value in the Last Bid field:

```
tcVar."Last Bid" = 32.25
```

The following statement uses **setFieldValue** to change the value in the Last Bid field:

```
tcVar.setFieldValue("Last Bid", 32.25)
```

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCFVAL;',0,"Defaultoverview",)} **Related Topics**

setFieldValue example

In the following example, the **pushButton** method for *correctName* locates a misspelled name in the Name field and uses **setFieldValue** to replace the original name:

```
; correctName::pushButton
method pushButton(var eventInfo Event)
var
  tc TCursor
  badName, goodName String
endVar


badName = "Usco"
goodName = "Unisco"
tc.open("Customer.db")
if tc.locate("Name", badName) then
  tc.edit()
  tc.setFieldValue("Name", goodName)    ; correct misspelled name
  tc.postRecord()                       ; post record to the table
  tc.endEdit()                          ; end Edit mode
  message("Usco replaced with Unisco.")
else                                    ; can't find "Usco" in Name field
  message("Can't find " + badName)
endIf
endMethod
```

setFlyAwayControl method

Specifies whether flyaway information is available to the **didFlyAway** method.

Syntax

```
setFlyAwayControl ( [ const yesNo Logical ] )
```

Description

setFlyAwayControl specifies in *yesNo* whether flyaway information is available to the **didFlyAway** method.

If you're working with indexed tables, the **didFlyAway**, **setFlyAwayControl**, and **unlockRecord** methods are closely related. When you call **unlockRecord**, the record is posted to the table (if no key violation exists) and moved to its sorted position. Depending on whether the record moved to a new position, the TCursor may not continue to point to the posted record. This behavior is called record flyaway.

You can use **didFlyAway** to determine whether the record did, in fact, fly away.

If **setFlyAwayControl** is set to Yes, Corel Paradox performs extra record-level checking for many operations. To maintain an application's speed set **setFlyAwayControl** to Yes only when the application needs flyaway information. By default, **setFlyAwayControl** is set it to No.

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCDIDFLYAWAY;OPAL_METH_TCPOSTRECORD;OPAL_METH_TCUNREC;',0,"Defaultoverview",)} Related Topics

setFlyAwayControl example

See the **didFlyAway** example.

setGenFilter method

Specifies conditions for including records in a TCursor.

Syntax

```
1. setGenFilter ( [ idxName String, [ tagName String, ] ] criteria DynArray[ ] AnyType ) Logical

2. setGenFilter ( [ idxName String, [ tagName String, ] ] criteria Array[ ] AnyType [ , fieldId Array[ ] AnyType ] ) Logical
```

Description

setGenFilter specifies conditions for including records in a TCursor. Records that meet all the specified conditions are included, records that don't are filtered out. Unlike **setRange**, this method does not require an indexed table. **setGenFilter** must be executed before opening a table using a TCursor.

In Syntax 1, a dynamic array (DynArray) named *criteria* specifies fields and filtering conditions. The index is the field name or number, and the item is the filter expression.

The following code specifies criteria based on the values of three fields:

```
criteriaDA[1]      = "Widget"            ; The value of the first field
                                         ; in the table is Widget.


criteriaDA["Size"] = "> 4"               ; The value of the field named
                                         ; Size is greater than 4.


criteriaDA["Cost"] = ">= 10.95, < 22.50" ; The value of the field named
                                         ; Cost is greater than or
                                         ; equal to 10.95  and less
                                         ; than 22.50.
```

If the DynArray is empty, all existing filter criteria are removed.

In Syntax 2, an Array named *criteria* specifies filtering conditions, and an optional Array named *fieldId* specifies field names and numbers. If you omit *fieldID*, conditions are applied to fields in the order they appear in the *criteria* array (the first condition applies to the first field, the second condition applies to the second field, and so on). The following example specifies the same criteria as the example for Syntax 1.

```
criteriaAR[1] = "Widget"
criteriaAR[2] = "> 4"
criteriaAR[3] = ">= 10.95, < 22.50"
fieldAR[1] = 1
fieldAR[2] = "Size"
fieldAR[3] = "Cost"
```

If the Array is empty, all existing filter criteria are removed.

For both syntaxes, *idxName* specifies an index name (Corel Paradox and dBASE tables) and *tagName* specifies a tag name (dBASE tables only). If you use these optional items, the index (and tag) are applied to the TCursor before the filtering criteria.

This method fails if the active record cannot be committed.

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCGETGENFILTER;OPAL_METH_TCDROPGENFILTER;OPAL_METH_TCSETRANGE;OPAL_METH_UISETGENFILTER;OPAL_INFO_TBUSINGRANGESANDFILTERS;',0,"Defaultoverview",)} Related Topics

setGenFilter example

In this example, the built-in **run** method for a script opens a TCursor onto the *Customer* table and sets filter criteria on the *State/Prov* field to equal CA. Then a **scan** loop is used to fill a dynamic array (DynArray) named *dynView* with the customer name and phone number. Finally, a **view** dialog box displays the data.

```
;Script :: run
method run(var eventInfo Event)
   var
      tc  TCursor
      dyn,
      dynView  DynArray[] AnyType
   endVar

   dyn["State/Prov"] = "CA"

   tc.open("CUSTOMER.DB")
   tc.setGenFilter(dyn)

   scan tc:
      dynView[tc."Name"] = tc."Phone"
   endScan

   dynView.view()
endMethod
```

setRange method

Specifies a range of records to associate with a Table variable. This method enhances the functionality of **setFilter**, which it replaces in this version. Code that calls **setFilter** continues to execute as before.

Syntax

```
1. setRange ( [ const exactMatchVal AnyType ] * [ , const minVal AnyType, const maxVal AnyType ] ) Logical

2. setRange ( rangeVals Array[ ] AnyType ) Logical
```

Description

setRange specifies conditions for including a range of records. Records that meet the conditions are included when the table is opened. **setRange** compares the criteria you specify with values in the corresponding fields of a table's index. If the table is not indexed, this method fails. If you call **setRange** without any arguments, the range criteria is reset to include the entire table.

Syntax 1 allows you to set a range based on the value of the first field of the index by specifying values in *minVal* and *maxVal*. For example, the following statement examines values in the first field of the index of each record:

```
tableVar.setRange(14, 88)
```

If a value is less than 14 or greater than 88, that record is filtered out. To specify an exact match on the first field of the index, assign the same value to *minVal* and *maxVal*. For example, the following statement filters out all values except 55:

```
tableVar.setRange(55, 55)
```

To set a range based on the values of more than one field, specify exact matches *except* for the last one in the list. For example, the following statement looks for exact matches on Corel and Corel Paradox (assuming they are the first fields in the index), and values ranging from 100 to 500 (inclusive) for the third field:

```
tableVar.setRange("Corel", "Corel Paradox", 100, 500)
```

In Syntax 2, you can pass an array of values to specify the range criteria, as listed in the following table.

Number of array items	Range specification
No items (empty array)	Resets range criteria to include the entire table
One item	Specifies a value for an exact match on the first field of the index
Two items	Specifies a range for the first field of the index
Three items	The first item specifies an exact match for the first field of the index; items 2 and 3 specify a range for the second field of the index.
More than three items	For an array of size n, specify exact matches on the first n-2 fields of the index. The last two array items specify a range for the n-1 field of the index.

Examples

setRange examples

Example1 Using **setRange** to specify fields for use in calculations

Example2 Using **setRange** with a criteria array containing more than three items

setRange example 1

The following example assumes that that the first field in Lineitem's key is Order No. and you want to know the total for order number 1005. When you press the *getDetailSum* button, the **pushButton** method opens a TCursor for *Lineitem* and limits the number of records included in the TCursor to those with 1005 in the first key field. After the call to **setRange**, this example uses **cSum** to display the sum of the Total field. Because the TCursor is pointing only to order number 1005, **cSum** reports summary information only for that order.

```
; getDetailSum::pushButton
method pushButton(var eventInfo Event)
var
  lineTC TCursor
  tblName String
endVar
tblName = "LineItem.db"
if lineTC.open(tblName) then


  ; this limits TCursor's view to records that have
  ; 1005 as their key value (Order No. 1005).
  lineTC.setRange(1005, 1005)


  ; now display the total for Order No. 1005
  msgInfo("Total for Order 1005", lineTC.cSum("Total"))
else
  msgStop("Sorry", "Can't open " + tblName + " table.")
endIf
endMethod
```

setRange example 2

The following example calls **setRange** using a criteria array that contains more than three items. The following code sets a range to include orders from a person with a specific first name, middle initial, and last name, and an order quantity ranging from 100 to 500 items. This code then counts the number of records in this range and displays the value in a dialog box. This example assumes that the *PartsOrd* table is indexed on the FirstName, MiddleInitial, LastName, and Qty fields.

```
; setQtyRange::pushButton
method pushButton(var eventInfo Event)
   var
      tcPartsOrd   TCursor
      arRangeInfo   Array[5] AnyType
      nuCount      Number
   endVar


   arRangeInfo[1] = "Frank"      ; FirstName (exact match)
   arRangeInfo[2] = "P."         ; MiddleInitial (exact match)
   arRangeInfo[3] = "Corel"    ; LastName (exact match)
   arRangeInfo[4] = 100          ; Minimum qty value
   arRangeInfo[5] = 500          ; Maximum qty value


   if tcPartsOrd.open("PartsOrd") then
         tcPartsOrd.setRange(arRangeInfo)
         nuCount = tcPartsOrd.cCount(1)
         nuCount.view("Number of big orders by Frank P. Corel:")
   else
         errorShow("Can't open the table.")
   endIf
endMethod
```

showDeleted method

Specifies whether to display deleted records in a dBASE table.

Syntax

showDeleted ([***yesNo***] **)** Logical

Description

showDeleted specifies whether to display deleted records in a dBASE table. You can use *yesNo* to specify Yes to display deleted records, or No if you don't want to display them. If omitted, *yesNo* is Yes by default.
showDeleted is valid only for dBASE tables because deleted records in a Corel Paradox table cannot be displayed.

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCCOMP;OPAL_METH_TCISRECORDDELETED;OPAL_METH_TCISSHOWDELETEDON;OPAL_METH_TCSHDEL;',0,"Defaultoverview",)} **Related Topics**

showDeleted example

In the following example, the **pushButton** method attached to *showDeletedRecs* calls **showDeleted** to display deleted records in ORDERS.DBF:

```
; showDeletedRecs::pushButton
method pushButton(var eventInfo Event)
var
  dbfTC TCursor
endVar
if dbfTC.open("Orders.dbf") then
  dbfTC.showDeleted(Yes)
else
  msgStop("Sorry", "Can't open Orders.dbf table.")
endIf
endMethod
```

skip method

Moves forward or backward a specified number of records in a table.

Syntax

```
skip ( [ const nRecords LongInt ] ) Logical
```

Description

skip Moves forward or backward a specified number of records in a table. If **skip** attempts to move beyond the limits of the table, an error is produced, and the active record will be the first or last record of the table. This operation fails if the active record cannot be committed (e.g., because of a key violation).

Positive values for *nRecords* move forward through the table (**skip**(1) is the same as **nextRecord**). Negative values move backward (**skip**(-1) is the same as **priorRecord**). A value of 0 doesn't move (**skip**(0) is the same as **currRecord**). If omitted, *nRecords* is 1 by default.

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCHOME;OPAL_METH_TCEND;OPAL_METH_TCNERE;OPAL_METH_TCPRRE;OPAL_METH_TCCURE;OPAL_METH_TCMTRE;',0,"Defaultoverview",)} **Related Topics**

skip example

The following example uses **skip** to change a TCursor's record position in a table:

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
  tc TCursor
endVar
tc.open("Orders.db")


tc.skip(5)     ; ahead 5 records. tc.recNo() = 6
tc.skip(-3)    ; back 3 records. tc.recNo() = 3
tc.skip(-5)    ; fails--attempted to move beyond the
               ; beginning of the table.
               ; tc.recNo() = 1
               ; tc.bot() = True


endMethod
```

sortTo method

Sorts a table.

Syntax

```
1. sortTo ( const destTable String, const numFields SmallInt, const sortFields Array[ ] String,
const sortOrder Array[ ] SmallInt ) Logical

2. sortTo ( const destTable Table, const numFields SmallInt, const sortFields Array[ ] String,
const sortOrder Array ) Logical
```

Description

sortTo sorts a table according to its field values, and saves the results in *destTable*.

sortFields is an array of strings or integers specifying which fields to sort. The size of the *sortFields* array is specified in *numFields*. *sortOrder* is an array of integers, where 0 specifies a sort in ascending order, and a value of 1 specifies descending order. The two arrays must be the same size, specified in *numFields*. Element 1 of *sortOrder* specifies how to sort the field named in element 1 of *sortFields*, and so on.

sortTo requires at least a read-only lock on the source table, and a full lock on the target table. If *destTable* already exists, it will be overwritten without asking for confirmation. If *destTable* is open, this method fails. You cannot use **sortTo** to sort a table onto itself; use a **sort** structure for that.

Example

 Related Topics

sortTo example

The following example sorts the *Customer* table to the CUSTSORT.DB table and opens the sorted table. If the *Customer* table cannot be write-locked, this example informs the user and aborts the operation. If the *CustSort* target table already exists, the user is prompted to continue or abort.

The following code goes in the Const window for the *sortCustButton* button:

```
; sortCustButton::Const
const
   kAscending = 0
   kDescending = 1
endConst
```

The following code is placed in the Var window for the *sortCustButton* button:

```
; sortCustButton::var
var
  sortFlds Array[2]  String
  sortOrder Array[2] SmallInt
  tc                 TCursor
  srcTbl, destTbl    String
  noSort             Logical
  sortTbl            TableView
endVar
```

The following code is attached to the button's **open** method. This code assigns **open** a TCursor for the *Customer* table and initializes the array elements. These assignments determine the sort criteria for **sortTo**:

```
; sortCustButton::pushButton
method open(var eventInfo Event)
srcTbl = "Customer.db"
destTbl = "CustSort.db"
if tc.open(srcTbl) then
  noSort = False                ; flag for pushButton method
  sortFlds[1] = "First Contact"  ; sort by First Contact
  sortOrder[1] = kAscending      ; in ascending order


  sortFlds[2] = "Country"        ; then by Country
  sortOrder[2] = kDescending     ; in descending order
else
  noSort = True
endIf


endMethod
```

The following code is attached to the **pushButton** method for the *sortCustButton* button. When the button is pressed, this code attempts to place a write lock on the source table (CUSTOMER.DB), asks the user if the target table exists (CUSTSORT.DB) and sorts *Customer* to *CustSort* based on the values in the *sortFlds* and *sortOrder* arrays. When CUSTSORT.DB is created or updated, this example opens it as a TableView.

```
; sortCustButton::pushButton
method pushButton(var eventInfo Event)
if noSort = False then
  if tc.lock("Write") then
    if isTable(destTbl) then
      if msgQuestion("Overwrite?",
           "Replace " + destTbl + " ?") <> "Yes" then
           msgInfo("Canceled", "Operation canceled.")
           return
      endIf
    endIf
    tc.sortTo(destTbl, 2, sortFlds, sortOrder)
    sortTbl.open(destTbl)
  else
    msgStop("Stop!", "Can't write-lock " + srcTbl + " table.")
  endIf
else
  msgStop("Sorry", "Can't open " + srcTbl + " table.")
endIf
endMethod
```

subtract method

Subtracts the records in one table from another table.

Syntax

```
1. subtract ( const destTable String ) Logical
2. subtract ( const destTable Table ) Logical
3. subtract ( const destTable TCursor ) Logical
```

Description

subtract determines whether records that reside in the source table also reside in *destTableName*. If matching records are found, **subtract** deletes them from *destTableName* without asking for confirmation.

If *destTableName* is keyed, **subtract** deletes the records with keys that match the values of key fields in the source table. If *destTableName* is not keyed, **subtract** deletes the records that match any record in the source table. Whether tables are keyed or not, this method considers only fields that *could* be keyed (based on data type, not position). For example, numeric fields are considered, but formatted memos are not. This method requires read/write access to both tables.

Throughout the retry period, this method attempts to place a full lock on both tables. If locks cannot be placed, an error results.

Note

- If the target table is not indexed, this operation can be time-consuming.

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCADD;OPAL_METH_TCCOPY;',0,"Defaultoverview",)}

Related Topics

subtract example

In the following example, the **pushButton** method for *subtractCust* deletes records from the *Customer* table that match those in the *Answer* table:

```
; subtractCust::pushButton
method pushButton(var eventInfo Event)
var
  ansTC, custTC TCursor
endVar


if ansTC.open(":PRIV:Answer.db") and
   custTC.open("Customer.db") then


  ansTC.subtract(custTC)              ; subtract Answer records from Customer


else
  msgStop("Stop!", "Can't open tables.")
endIf


endMethod
```

switchIndex method

Specifies an index to use to view a table's records.

Syntax

```
1. switchIndex ( [ const indexName String ] [ , const stayOnRecord Logical ] ) Logical

2. switchIndex ( [ const indexFileName String [ , const tagName String ] ] [ , const
stayOnRecord Logical ] ) Logical
```

Description

switchIndex specifies in *indexName* an index file to use to view a table. In Syntax 1, *indexName* specifies an index to use with a Corel Paradox table. If you omit *indexName*, the table's primary index is used.

Syntax 2 is for dBASE tables. *indexFileName* can specify an .NDX file or an .MDX file. The optional argument named *tagName* specifies an index tag in a production index (.MDX) file.

If the optional argument *stayOnRecord* is set to Yes in either syntax, this method maintains the active record after the index switch. If *stayOnRecord* is set to No (the default), the first record in the table becomes the active record.

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCREIND;OPAL_METH_TCREINAL;OPAL_METH_TBSIND;', 0,"Defaultoverview",)} Related Topics

switchIndex example

The following example assumes that *Customer* is a keyed Corel Paradox table that has a secondary index named NameAndState. This example opens a TCursor for *Customer*, calls **switchIndex** to switch from the primary index to the NameAndState index and displays the first value in the Name field. Because the TCursor is sorted on Name and State fields in ascending order, the field value displayed is the first name in ascending sort order.

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
   var
      tc TCursor
   endvar


   tc.open("Customer.db")            ; open TCursor for Customer
   tc.switchIndex("NameAndState")    ; switch to index NameAndState
   tc.home()                         ; move to the first record
   msgInfo("First Record", tc.Name)  ; display value in Name field
   tc.switchIndex( )                 ; to restore primary index
   { tc.switchIndex (" ", True) to stay on the same record. }
   msgInfo("First Record", tc.Name)  ; display value in Name field
endMethod
```

tableName method

Returns the name of the table associated with a TCursor.

Syntax

```
tableName ( ) String
```

Description

tableName returns the name of the table associated with a TCursor. This method is used to pass variables to the TCursor **open** method.

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCTRIG;',0,"Defaultoverview",)} Related Topics

tableName example

In the following example, the **pushButton** method for *thisButton* uses the **findFirst** and **findNext** methods from the FileSystem type to locate Corel Paradox tables in the working directory. This code searches each table for a value in the Name field of the current table. This example opens all of the tables in the current directory that have Unisco in the Name field:

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
  fs FileSystem
  tc TCursor
  tb TableView
endVar
if fs.findFirst("*.db") then
  while fs.findNext()
    tc.open(fs.Name())                  ; open TCursor for a .db file
    if tc.locate("Name", "Unisco") then ; if we find Unisco in Name field
      tb.open(tc.tableName())           ; open table associated with TCursor
    endIf
    tc.close()
  endWhile
endIf


endMethod
```

tableRights method

Specifies whether the user has the right to perform table operations.

Syntax

```
tableRights ( const rights String ) Logical
```

Description

tableRights specifies whether the user has the right to perform table operations. The following table describes *rights*:

Value	Description
ReadOnly	Specifies the right to read from the table without making changes
Modify	Specifies the right to enter or change data
Insert	Specifies the right to add new records
InsDel	Specifies the right to add and delete records
Full or All	Specifies the right to perform all of the above operations

This method returns True if the user has the specified rights; otherwise, it returns False.

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCFRIG;',0,"Defaultoverview",)} Related Topics

tableRights example

The following example reports whether the user has InsDel rights to the *Orders* table:

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
   myRights Logical
   ordersTC TCursor
endVar
ordersTC.open("orders.db")
ordersTC.edit()
myRights = ordersTC.tableRights("InsDel")

  ; this displays True if you have InsDel rights to Orders.db
msgInfo("Rights to Enter?", myRights)

endMethod
```

type method

Returns a table's type.

Syntax

```
type ( ) String
```

Description

type returns the string value COREL PARADOX or DBASE to specify the table's type.

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCISASSIGNED;OPAL_METH_TCTNAM;',0,"Defaultoverview",)} Related Topics

type example

The following example removes deleted records from the *Orders* table if **type** returns DBASE. If **type** returns Corel Paradox, a message is displayed:

```
; compact::pushButton
method pushButton(var eventInfo Event)
var
  tc TCursor
endVar


tc.open("Orders.db")


; if Orders.db is a dBASE table
if tc.type() = "dBASE" then
  ; remove deleted records
  tc.compact()
else
  ; otherwise, display the type of table
  msgStop("Stop!", "Orders.db is a " + tc.type() + " table.")
endIf

endMethod
```

unDeleteRecord method

Undeletes the active record from a dBASE table.

Syntax

unDeleteRecord ( ) Logical

Description

unDeleteRecord undeletes the active record from a dBASE table. This operation is successful only if **showDeleted** is set to True, the active record is a deleted record, and the TCursor is in Edit mode.

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCDREC;OPAL_METH_TCISRECORDDELETED;OPAL_METH_TCISSHOWDELETEDON;OPAL_METH_TCSHDEL;',0,"Defaultoverview",)} **Related Topics**

unDeleteRecord example

The following example opens a TCursor for SCORES.DBF (a dBASE table) and uses **showDeleted** to display deleted records. This code then attempts to locate a specific record in the table. **isRecordDeleted** determines whether the record has been deleted; if it has, it is undeleted using **unDeleteRecord**. The following code is attached to the **pushButton** method for *thisButton*:

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
  tc TCursor
endVar
tc.open("Scores.dbf")                ; open TCursor on a dBASE table
tc.showDeleted()                     ; show deleted records
if tc.locate("Name", "Jones") then   ; if locate finds Jones in Name field
  if tc.isRecordDeleted() then       ; if the record has been deleted
    tc.edit()                        ; begin Edit mode
    tc.undeleteRecord()              ; undelete the record
    message("Jones record undeleted")
  endIf
else
  msgStop("Error", "Can't find Jones.")
endIf
endMethod
```

unlock method

Unlocks a specified table that is pointed to by TCursor.

Syntax

```
unlock ( const lockType String ) Logical
```

Description

unlock attempts to remove locks explicitly placed on the table pointed to by a TCursor. *lockType* is one of the following String values, listed in order of decreasing strength and increasing concurrency:

String value	Description
Full	The current session has exclusive access to the table. No other session can open the table. Cannot be used with dBASE tables.
Write	The current session can write to and read from the table. No other session can place a write lock or a read lock on the table.
Read	The current session can read from the table. No other session can place a write lock, full lock, or exclusive lock on the table.

unlock removes locks that have been explicitly placed by a particular user or application using **lock**. **unlock** has no effect on locks placed automatically by Corel Paradox. To ensure maximum concurrent availability of tables unlock a table that has been explicitly locked as soon as the lock is no longer needed. If you lock a table twice, you must unlock it twice. You can use **lockStatus** (defined for the TCursor and UIObject types) to determine how many explicit locks you have placed on a table. If you try to unlock a table that isn't locked or cannot be unlocked, **unlock** returns False .

If successful, this method returns True; otherwise, it returns False.

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCLOCK;OPAL_METH_TCLSTA;OPAL_METH_UILOCKSTATUS;',0,"Defaultoverview",)} Related Topics

unlock example

The following example opens a TCursor for *Customer* (a Corel Paradox table), places a full lock on the table and uses **reIndex** to rebuild the *Phone_Zip* index. Once the index is rebuilt, this code unlocks *Customer* so other users on a network can gain access to the table:

```
; reindexCust::pushButton
method pushButton(var eventInfo Event)
var
  tc TCursor
  pdoxTbl String
endVar
pdoxTbl = "Customer.db"


if tc.open(pdoxTbl) then
  if tc.lock("Full") then    ; attempt to gain exclusive access
    tc.reIndex("Phone_Zip")  ; rebuild Phone_Zip index
    tc.unLock("Full")        ; unlock the table
  else
    msgStop("Sorry", "Can't lock " + pdoxTbl + " table.")
  endIf
else
  msgStop("Sorry", "Can't open " + pdoxTbl + " table.")
endIf
endMethod
```

unLockRecord method

Unlocks the active record.

Syntax

unLockRecord ( ) Logical

Description

unLockRecord unlocks the active record. If you attempt to unlock a record that isn't locked, you'll get an error. This operation fails if the active record cannot be committed (e.g., because of a key violation).

If the table containing the record is indexed, the record is posted to the table and moved to its sorted position. Depending on whether the record moved to a new position, the TCursor may not continue to point to the posted record. This behavior is referred to as record fly away.

If a key value changes when the record is unlocked, the record may fly away to a new position in the table; however, the TCursor does not fly with it. You can also use **didFlyAway** to determine whether the record did fly away.

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCDIDFLYAWAY;OPAL_METH_TCLREC;OPAL_METH_TCPOSTRECORD;OPAL_METH_TCSETFLYAWAYCONTROL;',0,"Defaultoverview",)} **Related Topics**

unLockRecord example

In the following example, the **pushButton** method for *thisButton* attempts to locate a misspelled value in the *Name* field of the *Customer* table. If the value is found, this code locks the record, corrects the value in the field and unlocks the record using **unLockRecord**:

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
  tc TCursor
endVar
if tc.open("Customer.db") then
  if tc.locate("Name", "Usco") then
    tc.edit()
    tc.lockRecord()           ; lock active record
    tc.Name = "Unisco"        ; change field value
    tc.unlockRecord()         ; unlock active record
    message("Name changed to \"Unisco\"")
  else
    msgStop("Sorry", "Can't find \"Usco\" in \"Name\" field.")
  endIf
else
  msgStop("Sorry", "Can't open Customer.db table.")
endIf


endMethod
```

update method

Assigns values to fields in the active record of a TCursor.

Syntax

```
1. update ( const fieldName String, const fieldValue AnyType
       [ , const fieldName String, const fieldValue AnyType ] * ) Logical
2. update ( const fieldNum SmallInt, const fieldValue AnyType
       [ , const fieldNum SmallInt, const fieldValue AnyType ] * ) Logical
```

Description

update assigns values to one or more fields in the active record of a TCursor. **update** allows you to update an entire record using a single statement instead of assigning field values one at a time. Use *fieldName* (Syntax 1) or *fieldNum* (Syntax 2) to specify fields. Use *fieldValue* in Syntax 2 to specify the new field value.

You can also combine field names and field numbers in the same update statement. Performance improves if you use field numbers instead of field names.

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCSFVA;',0,"Defaultoverview",)} **Related Topics**

update example

The following example uses **update** to set the values of three fields using one statement.
First, the following code assigns values to the *PartNum*, *PartName*, and *Cost* fields of the *Parts* table without using *update*:

```
var
   partsTC   TCursor
   partNumID   SmallInt
endVar


partsTC.open("parts")
partNumID = partsTC.fieldNo("PartNum")


if partsTC.locate("PartName", "Widget") then
   partsTC.edit()


   partsTC.(partNumID) = "G01"
   partsTC.PartName    = "Gadget"
   partsTC.Cost        = 2.50


   partsTC.endEdit()
endIf
```

The following code calls **update** to accomplish the same thing:

```
var
   partsTC   TCursor
   partNumID   SmallInt
endVar


partsTC.open("parts")
partNumID = partsTC.fieldNo("PartNum")


if partsTC.locate("PartName", "Widget") then
   partsTC.edit()


   partsTC.update(partNumID, "G01", "PartName", "Gadget", "Cost", 2.50)


   partsTC.endEdit()
endIf
```

updateRecord method

Updates an existing record with data from a new record when a key violation exists.

Syntax

```
updateRecord ( [ const moveTo Logical ] ) Logical
```

Description

updateRecord overwrites an existing record with values from an unposted, new record when a key violation exists. The record is posted to the table. If an optional argument *moveTo* is False, the TCursor points to the record after it is posted to the table; if True, the TCursor points to the record following the position of the original record.

If no key violation exists, this method behaves like **unlockRecord**.

Example

{button ,AL(`OPAL_TYPE_TCURSOR;OPAL_METH_TCATTACHTOKEYVIOL;OPAL_METH_TCLREC;OPAL_METH_TCPOSTRECORD;OPAL_METH_TCUNREC;',0,"Defaultoverview",)} Related Topics

updateRecord example

See the **attachToKeyViol** example.

TextStream type

A TextStream is a sequence of characters read from or written to a text file. TextStreams contain ANSI characters only—formatting information such as font, alignment, and margins is not included. TextStreams also contains non-printing characters (e.g., carriage returns and line feeds (CR/LF)).

Corel Paradox maintains a file position cursor that behaves like an insertion point cursor in a word processor. The cursor tells you how far (how many characters) you are from the beginning of the file. Counting begins with 1 (not with 0, as in some other languages).

Methods for the TextStream type

advMatch
close
commit
create
end
eof
home
open
position
readChars
readLine
setPosition
size
writeLine
writeString

Print related ObjectPAL methods and examples

advMatch method

Searches for a pattern of characters in a text file.

Syntax

```
advMatch ( var startIndex LongInt, var endIndex LongInt, const pattern String ) Logical
```

Description

advMatch searches a text file for a pattern of characters specified by *pattern*. If *startIndex* is assigned a value, the search begins at the *startIndex* position; otherwise, the search begins at the beginning of the file. The position in *endIndex* does not indicate the end of the range to search. If the pattern is found, the position of the first matching character is stored in *startIndex*, and the position of the last matching character is stored in *endIndex.*

advMatch returns True if *pattern* is found in the file; otherwise, it returns False. By default, this method is case sensitive but you can use the String procedure **ignoreCaseInStringCompares** to change the case behavior.

If you supply *pattern* from within a method, you must use two backslashes when you want to tell **advMatch** to treat a special character as a literal; for example, \\(tells **advMatch** to treat the parenthesis as a literal character. Two backslashes are required in this situation because the compiler and **advMatch** understand backslashes differently. When the compiler sees a string with an embedded escape sequence (e.g., a \tstart), it interprets the \t as a tab, followed by the word start. The backslash character has a special meaning to the compiler, but it also has a special meaning to **advMatch**. For more information, see the entry for **advMatch** in the String type.

If you supply *pattern* from a field in a table or a TextStream, special **advMatch** symbols are recognized without a backslash, and one backslash and plus symbol (\+) yields a literal character.

To specify *pattern*, use a string with the following optional symbols:

Symbol	Matches
\	Includes special characters as regular characters (e.g., \t for Tab). Use two backslashes in quoted strings.
[ ]	Match the enclosed set. (e.g., [aeiou0-9] match a, e, i, o, u, and 0 through 9)
[^]	Does not match the enclosed set. (e.g., [^aeiou0-9] matches anything except a, e, i, o, u, and 0 through 9)
()	Specifies grouping
^	Specifies the beginning of string
$	Specifies the end of string
..	Matches anything
@	Matches any single character
*	Specifies zero or more of the preceding character or expression
+	Specifies one or more of the preceding character or expression
?	Specifies none or one of the preceding character or expression
|	Specifies OR operation

For examples, see Sample search strings with wildcards in the User's Guide help.

Example

{button ,AL(`OPAL_TYPE_TEXTSTREAM;OPAL_METH_TSHOME;OPAL_METH_TSEND;OPAL_METH_TSSPOS;OPAL_METH_TSPOSI;OPAL_METH_STAMAT;OPAL_METH_STIGN;',0,"Defaultoverview",)} **Related Topics**

advMatch example

The following example assumes that a file named PDXQUOTE.TXT exists in the working directory. The file contains the following text:

```
How wonderful that we have met with Corel Paradox.
Now we have some hope of making progress.
Niels Bohr
```

The call to **advMatch** specifies @o@e as the pattern to search. This pattern matches any character, followed by an **o** followed by any character followed by an **e**. If the specified pattern is found, the variables *firstChar* and *lastChar* store the positions of the first and last matching characters. The calls to **setPosition** and **readChars** read the matching characters and store them in the variable *theMatch*.

```
; findSome::pushButton
method pushButton(var eventInfo Event)
var
  pdq                TextStream
  firstChar, lastChar LongInt
  theMatch           String
endvar
if pdq.open("pdxquote.txt", "R") then
  if pdq.advMatch(firstChar, lastChar, "@o@e") then
    msgInfo("The position found", firstChar)
    pdq.setPosition(firstChar)
    pdq.readChars(theMatch, lastChar - firstChar)
    message(theMatch)              ; displays "some"
  else
    msgInfo("Sorry", "Match not found.")
  endIf
  pdq.close()
else
  msgInfo("Sorry", "Couldn't open the requested text file.")
endIf
endMethod
```

close method

Closes a text file.

Syntax

```
close ( ) Logical
```

Description

close closes a text file and writes the contents of all text buffers to a disk. **close** also ends the association between a TextStream variable and the underlying text file.

Example

{button ,AL(`OPAL_TYPE_TEXTSTREAM;OPAL_METH_TSOPEN;OPAL_METH_TSCOMM;',0,"Defaultoverview",)} Related Topics

close example

The following example declares a TextStream variable named *ts*, and calls **open** to associate *ts* with the text file named PDXQUOTE.TXT. The code then calls **close** to end the association.

```
; quoteALine::pushButton
method pushButton(var eventInfo Event)
var
  ts        TextStream
  firstLine String
endvar
ts.open("pdxQuote.txt", "R")
ts.readLine(firstLine)
firstLine.view("Line 1 of PDXQUOTE.TXT")
ts.close()
endMethod
```

commit method

Writes the contents of the text buffer to a disk.

Syntax

```
commit ( )
```

Description

commit empties the text buffer and writes the contents to a disk. The file remains open and the cursor position does not change.

Example

{button ,AL(`OPAL_TYPE_TEXTSTREAM;OPAL_METH_TSWRST;',0,"Defaultoverview",)} **Related Topics**

commit example

In the following example, the *createText* button creates a new file named MYTEXT.TXT, adds a line to it, commits the current version of the TextStream and closes the file:

```
; createText::pushButton
method pushButton(var eventInfo Event)
var
  ts TextStream
endVar


ts.create("myText.txt")
msgInfo("TextStream position is now", ts.position()) ; displays 1


ts.writeLine("This is some text.")
msgInfo("TextStream position is now", ts.position()) ; displays 21


ts.commit()
msgInfo("TextStream position is now", ts.position()) ; still 21
ts.close()


endMethod
```

create method

Creates a text file for reading and writing.

Syntax

```
create ( const fileName String ) Logical
```

Description

create creates the text file specified by *fileName* and opens it for reading and writing. If *fileName* already exists, **create** overwrites it without asking for confirmation. You can specify where to create the file using a full DOS path or an alias. If you don't specify a path or alias, Corel Paradox creates the file in the working directory.

This method returns True if successful; otherwise, it returns False. If the file is successfully created, it is opened for reading and writing.

Note

- The following statements are equivalent:

```
ts.create("newText.txt")
ts.open("newText.txt, "NW")
```

Example

{button ,AL(`OPAL_TYPE_TEXTSTREAM;OPAL_METH_TSOPEN;OPAL_METH_TSCLOS;',0,"Defaultoverview",)} Related Topics

create example

In the following example, code is attached to a button's **pushButton** method. It consists of a variable declaration block, a procedure declaration, and the body of the method. In the body of the method, the **findFirst** determines whether a file named RICK.TXT exists. If it doesn't exist, a custom procedure named **addLine** creates it and adds a line to it. If the file does exist, a dialog box confirms the decision to overwrite the file.

```
; createFile::pushButton
var
  ts               TextStream
  firstLine        String
  allLines Array[] String
  fs               FileSystem
endvar


proc addLine()
; Create a file, open for writing and reading
  ts.create(":PRIV:rick.txt")
  ts.writeLine("Here's looking at you, kid.")
  ts.home()
  ts.readLine(allLines)
  allLines.view("Rick says:")
endProc


method pushButton(var eventInfo Event)
if not fs.findFirst(":PRIV:rick.txt") then
  addLine()
else
  if msgYesNoCancel(":PRIV:RICK.TXT",
                    "Overwrite this file?") = "Yes" then
    addLine()
  endIf
endIf
endMethod
```

end method

Sets the cursor to the end in a text file.

Syntax

```
end ( )
```

Description

end sets the cursor to the last character in a text file.

Example

{button ,AL(`OPAL_TYPE_TEXTSTREAM;OPAL_METH_TSHOME;OPAL_METH_TSSPOS;OPAL_METH_TSEOF;', 0,"Defaultoverview",)} Related Topics

end example

The following example assumes that a file named PDXQUOTE.TXT is stored in the private directory. The file contains the following text:

```
How wonderful that we have met with Corel Paradox.
Now we have some hope of making progress.
Niels Bohr
```

The following code is attached to the built-in **newValue** method of a field object. The field object displays two radio buttons with the values Overwrite and Append. Choose one overwrite the file or append information to the end of the file. If you choose Overwrite, the call to **home** moves the cursor to position 1. If you choose Append, the call to **end** moves the cursor to the end of the file.

```
; insertAppendField::changeValue
method newValue(var eventInfo Event)
var
  ts TextStream
  allLines Array[] String
endVar
if eventInfo.reason() = EditValue then
  ts.open(":PRIV:pdxquote.txt", "W")
  switch
    case self.value = "Overwrite" :
      ts.home()
      ts.writeLine(DateTime())   ; time stamp the file at beginning
      ; file will read:
      ; DateTimeStamp (depends on date/time)
      ; have met with Corel Paradox.
      ; Now we have some hope of making progress.
      ; Niels Bohr
    case self.value = "Append" :
      ts.end()
      ts.writeLine(DateTime())   ; time stamp the file at end
      ; file will read:
      ; How wonderful that we have met with Corel Paradox.
      ; Now we have some hope of making progress.
      ; Niels Bohr
      ; DateTimeStamp (depends on date/time)
  endSwitch
  ts.home()
  ts.readLine(allLines)
  allLines.view()
  ts.close()
endIf
endMethod
```

eof method

Determines whether the cursor attempts to move past the end of a text file.

Syntax

```
eof ( ) Logical
```

Description

eof returns True if the cursor attempts to move past the end of a text file; otherwise, it returns False.

Example

eof example

The following example assumes that a file named PDXQUOTE.TXT resides in the private directory. The file contains the following text:

```
How wonderful that we have met with Corel Paradox.
Now we have some hope of making progress.
Niels Bohr
```

The **while** loop reads each of the three lines from the file and displays them in a dialog box. **eof** displays a dialog box telling the user that there's no more text in the file.

```
; lineAtATime::pushButton
method pushButton(var eventInfo Event)
var
  pdq      TextStream
  textLine String
endVar


pdq.open(":PRIV:pdxquote.txt", "r")
while not pdq.eof()        ; quit loop when you hit the end of the file
  pdq.readLine(textLine)   ; read the next line
  msgInfo("Position " + String(pdq.position()), textLine)
endWhile
msgInfo("Finished", "No more text")
endMethod
```

home method

Sets the cursor to the beginning of a text file.

Syntax

```
home ( )
```

Description

home sets the cursor to the first character of a text file.

Example

{button ,AL(`OPAL_TYPE_TEXTSTREAM;OPAL_METH_TSEND;OPAL_METH_TSEOF;OPAL_METH_TSSPOS;',0, "Defaultoverview",)} Related Topics

home example

See the **end** example.

open method

Opens a text file in a specified mode.

Syntax

```
open ( const fileName String, const mode String ) Logical
```

Description

open opens a text file named *fileName* in the mode specified by *mode*. **open** then and associates a FileSystem variable with the underlying file. Mode specifications are case-insensitive. The following table displays mode specifications:

Mode specification	Description
a	Append and read
r	Read only
w	Write and read
nw	New file (write and read)

If the file exists, the nw mode overwrites the file without asking for confirmation.

Note

- The following statements are equivalent:
  ```
  ts.open("new.txt", "NW")
  ts.create("new.txt")
  ```

If you open a file in any r, w, or nw modes, the cursor moves to the beginning of the file.

You can specify a directory from which to open the file using a full DOS path or an alias. If you don't specify a path or an alias, Corel Paradox searches for the file in the working directory.

This method returns True if successful; otherwise, it returns False.

Examples

{button ,AL(`OPAL_TYPE_TEXTSTREAM;OPAL_METH_TSCREA;OPAL_METH_TSCLOS;',0,"Defaultoverview",)} Related Topics

open method examples

Example1 Using an alias with **open**

Example2 Using **open** with two TextStream variables

open example 1

The following example uses an alias with **open** to create a text file in the private directory, and write a line of text to it:

```
var
   ts TextStream
endVar
if ts.open(":PRIV:memo14.txt", "NW") then
   ts.writeLine("This is private!")
endIf
```

open example 2

The following example declares two TextStream variables (*ts1* and *ts2*) and calls **open** to associate each of them with a text file named NEWTEXT.TXT. Both variables have equal rights to the file, and Corel Paradox maintains separate cursors for each variable. As statements are written to the file, messages display the cursor position for each variable.

```
; openStreams::pushButton
method pushButton(var eventInfo Event)
var
  ts1, ts2  TextStream
  firstLine String
  allLines  Array[] String
endvar
ts1.open("newText.txt", "nw")      ; open a new file read/write
ts1.writeLine("Written by ts1.")
ts1.writeLine("This is line 2.")
msgInfo("Text stream one", ts1.position()) ; displays 35
ts1.commit()                       ; write it out to disk, so that
                                   ; ts2 will get most current version


ts2.open("newText.txt", "w")       ; open existing file read/write
msgInfo("Text stream one", ts1.position()) ; displays 35
msgInfo("Text stream two", ts2.position()) ; displays 1


ts2.writeLine("Written by ts2.")
msgInfo("Text stream one", ts1.position()) ; displays 35
msgInfo("Text stream two", ts2.position()) ; displays 18


ts1.home()
ts1.readLine(allLines)     ; reads all lines into an array
allLines.view("ts1")       ; displays:
                           ; Written by ts1.
                           ; This is line 2.
; ts1 does not reflect changes made by ts2
; unless ts1 is closed and reopened.
endMethod
```

position method

Returns the cursor's position in a text file.

Syntax

```
position ( ) LongInt
```

Description

position returns an integer representing the cursor's position in a text file. **position** counts both printing and non-printing characters, beginning with 1 (not with 0).

It may be helpful to think of **position** as returning the number of the next character in the file.

Example

{button ,AL(`OPAL_TYPE_TEXTSTREAM;OPAL_METH_TSSPOS;OPAL_METH_TSSIZ;',0,"Defaultoverview",)}

Related Topics

position example

The following example creates a new text file and calls **position**. It returns 1. The call to **writeLine** adds 14 characters to the file: 12 printing characters and the carriage return and line feed (CR/LF) pair. The next character will be 15, so **position** returns 15.

```
var newFile TextStream endVar
newFile.open("newmemo.txt", "nw")
message(newFile.position()) ; displays 1
sleep(1000)
newFile.writeLine("Don't panic.")
message(newFile.position()) ; displays 15
                            ; 12 printing characters + CR/LF = 14
                            ; next character will be 15
sleep(1000)
```

readChars method

Reads a specified number of characters in a text file.

Syntax

```
readChars ( var string String, const nChars SmallInt ) Logical
```

Description

readChars reads the number of characters specified in *nChars* and stores them in *string*. **readChars** begins at the current cursor position and returns True if successful; otherwise, it returns False.

Example

{button ,AL(`OPAL_TYPE_TEXTSTREAM;OPAL_METH_TSRELN;OPAL_METH_TSSPOS;OPAL_EX_TSAMAT;',0,"Defaultoverview",)} **Related Topics**

readChars example

The following example assumes that a file named PDXQUOTE.TXT resides in the working directory. The file contains the following text:

```
How wonderful that we have met with Corel Paradox.
Now we have some hope of making progress.
Niels Bohr
```

The following code calls **readChars** to read the first 100 characters in the file:

```
; getLetters::pushButton
method pushButton(var eventInfo Event)
var
  letter  TextStream
  myChars String
endVar
letter.open("pdxquote.txt", "r")
if letter.readChars(myChars, 100) then

  msgInfo("The first 100 characters are:", myChars)
endIf
endMethod
```

readLine method

Reads the characters in a line of text.

Syntax

```
1. readLine ( var value String ) Logical
2. readLine ( var stringArray Array[ ] String ) Logical
```

Description

readLine reads the characters in a line of text until it encounters a CR/LF pair (or 32,767 characters have been read). **readLine** then moves the cursor to the first position after the CR/LF pair (or after the 32,767th character). **readLine** begins reading from the current cursor position. This method returns True if successful; otherwise, it returns False.

Syntax 1 stores a single line in *value* (the CR/LF pair is not stored).

Syntax 2 stores the entire file in *stringArray. stringArray* is a resizeable array of strings and each array item stores one line from the file (the CR/LF pair is not stored).

 **Examples**

{button ,AL(`OPAL_TYPE_TEXTSTREAM;OPAL_METH_TSREADCHARS;OPAL_METH_TSWRLN;',0,"Defaultoverview",)} Related Topics

readLine method examples

Example1	Reading the first line of the file into a string variable
Example2	Reading the entire file into an array

readLine example 1

The following example creates a two-line text file and calls **readLine** to read the first line into a String variable. **readLine** reads four characters in the first line, skips over the CR/LF characters, and sets the cursor.

```
method pushButton(var eventInfo Event)
var
   ts TextStream
   oneLine String
endvar


ts.create("newtext.txt")
ts.writeLine("1234")
ts.writeLine("5678")
ts.home()


ts.readLine(oneLine)
message(oneLine.size()) ; displays 4 (doesn't include CR/LF)
sleep(1000)
message(ts.position()) ; displays 7 (skips over CR/LF)
sleep(1000)
endMethod
```

readLine example 2

The following example creates a three-line text file and calls **readLine** to read the entire file into an array. The array is displayed in a dialog box.

```
method pushButton(var eventInfo Event)
var
   letter TextStream
   allLines Array[] String
endVar


letter.open("letter.txt", "nw")
letter.writeLine("Dear Customer,")
letter.writeLine("Thank you for your interest in our new product.")
letter.writeLine("A representative will call you next week.")


letter.home() ; move the cursor to the beginning of the file


letter.readLine(allLines)
allLines.view("Entire letter")     ; displays the entire letter
letter.close()
endMethod
```

setPosition method

Sets the cursor position in a text file.

Syntax

```
setPosition ( const offset LongInt )
```

Description

setPosition sets the cursor position in a text file. *offset* specifies the cursor's position from the beginning of a text file. CR/LF characters are considered part of the file and can be overwritten.

Examples

{button ,AL(`OPAL_TYPE_TEXTSTREAM;OPAL_METH_TSPOSI;OPAL_METH_TSSIZ;OPAL_METH_TSEOF;',0,"Defaultoverview",)} Related Topics

setPosition method examples

Example1 Using **setPosition** to display specific lines

Example2 Moving the cursor outside a file

setPosition example 1

In the following example, the *showPositions* button writes a line to a new text file, MEMO.TXT. The code then moves back to the fourth character, overwrites that character with the number 4, and displays the line.

```
; showPositions::pushButton
method pushButton(var eventInfo Event)
var
  myFile  TextStream
  lineOne String
endVar
myFile.open(":PRIV:memo.txt", "nw")        ; open new file as read/write
myFile.writeLine("1235")                   ; 4 characters plus CR/LF
msgInfo("Where am I?", myFile.position())  ; displays 7


myFile.setPosition(4)                ; move to character 4
myFile.writeString("4")              ; now, line is "1234"
myFile.home()                        ; same as setPosition(1)
myFile.readLine(lineOne)
msgInfo("This is line one", lineOne) ; displays  "1234"
endMethod
```

setPosition example 2

The following example shows what happens when you attempt to move the cursor beyond the end of a file or before the beginning of a file.

```
; showPositions::pushButton
method pushButton(var eventInfo Event)
var
  myFile  TextStream
endVar


myFile.open(":PRIV:memo.txt", "r")  ; open existing file for read
myFile.setPosition(100)             ; beyond end of file
msgInfo("End", myFile.position())   ; displays 7 the real end
myFile.setPosition(-100)            ; before beginning of file
msgInfo("Home", myFile.position())  ; displays 1
 the beginning
endMethod
```

size method

Returns the number of characters in a text file.

Syntax

size ( ) LongInt

Description

size returns the number of characters in a text file, including non-printing characters (e.g., carriage returns and line feeds).

Example

{button ,AL(`OPAL_TYPE_TEXTSTREAM;OPAL_METH_TSPOSI;OPAL_METH_TSSPOS;OPAL_METH_TSEOF;',0, "Defaultoverview",)} Related Topics

size example

The following example creates a TextStream, writes a line to it and displays the file's size.

```
; showSize::pushButton
method pushButton(var eventInfo Event)
var
  myText TextStream
endVar
myText.create("short.txt")
myText.writeLine("1234")
msgInfo("What size am I?", myText.size()) ; displays 6
; 4 printing characters "1234", and 2 nonprinting characters CR/LF
myText.close()
endMethod
```

writeLine method

Writes a string to a text file.

Syntax

```
writeLine ( const value AnyType [ , const value AnyType ] * ) Logical
```

Description

writeLine writes a comma-separated list of *value*s to a text file and appends a CR/LF character pair. Compare this method to **writeString**, which doesn't append a CR/LF pair.

Example

{button ,AL(`OPAL_TYPE_TEXTSTREAM;OPAL_METH_TSRELN;OPAL_METH_TSWRST;',0,"Defaultoverview",)} Related Topics

writeLine example

See the **create** example.

writeString method

Writes a character string to a text file.

Syntax

```
writeString ( const value AnyType, [ , const value AnyType ] * ) Logical
```

Description

writeString writes a comma-separated list of *values* to a text file, but does not append a CR/LF pair. Compare this method to **writeLine**, which does append a CR/LF pair.

Example

{button ,AL(`OPAL_TYPE_TEXTSTREAM;OPAL_METH_TSWRLN;',0,"Defaultoverview",)} Related Topics

writeString example

The following example assigns strings to the variables *lo* and *hi* and uses **writeString** to write them to an open TextStream.

```
; goodAdvice::pushButton
method pushButton(var eventInfo Event)
var
   myText TextStream
   lo, hi String
endVar
lo = "Buy low. "
hi = "Sell high."
myText.open(":PRIV:advice.txt", "nw")         ; open a new file
myText.writeString(lo, hi)
msgInfo("File size:", string(myText.size()))  ; displays 19
; Buy low. = 9 characters, Sell High. = 10 characters. 10 + 9 = 19.
myText.close()
endMethod
```

Time type

Time variables store times in hour-minute-second-millisecond format. The following characters can be used as separators: blank, tab, space, comma (,), hyphen (-), slash (/), period (.), colon (:), and semicolon (;). Time values must be enclosed in quotation marks.

Time values must be explicitly declared. For example, the following statements assign a time of 10 minutes and 40 seconds past eleven o'clock in the morning to the Time variable *ti* :

```
var ti Time endVar
ti = Time("11:10:40 am")
```

Valid time values are determined by the current Corel Paradox time format. If the current time format is set to a 12-hour format (e.g., hh:mm:ss), Time type methods consider hh:mm:ss to be a valid time format. Use **formatSetTimeDefault** procedure defined for the System type to set Corel Paradox time formats with ObjectPAL.

The Time type includes several derived methods from the DateTime and AnyType types.

Methods for the Time type

AnyType	DateTime	**Time**
blank	hour	**time**
dataType	milliSec	
isAssigned	minute	
isBlank	second	
isFixedType		
view		

Print related ObjectPAL methods and examples

time procedure

Casts a value as a time or returns the current time.

Syntax

```
time ( [ const value AnyType ] ) Time
```

Description

time casts *value* as a time or returns the current time according to the system clock. *value*, if specified, must match the current Corel Paradox time format. For more information, see to the System type procedure **formatSetTimeDefault**.

Examples

{button ,AL(`OPAL_TYPE_TIME;OPAL_METH_DTHOUR;OPAL_METH_DTMILLISEC;OPAL_METH_DTMINUTE;OPAL_METH_DTSECOND;',0,"Defaultoverview",)} Related Topics

time procedure examples

Example1 Converting a string value into a time value

Example2 Using code attached to a button's **pushButton** method

time example 1

The following example calls **time** to convert a string value to a time value:

```
var
   st String
   ti Time
endVar


st = "12:21:33 am"
ti = time(st)
```

time example 2

The following example displays the current time in a dialog box. The display format varies according to the user's current time format. This code is attached to a button's **pushButton** method:

```
; timeButton::pushButton
method pushButton(var eventInfo Event)


  ; displays the current time in a dialog box
   msgInfo("Current Time", time())


endMethod
```

UIObject type

UIObjects (user interface objects) create the user interface for an application. Anything you can place in a form or report is a UIObject. UIObjects include bands, bitmaps, boxes, buttons, cells, charts, crosstabs, ellipses, field objects, forms, groups, lines, lists, multi-record objects, OLE objects, pages, record objects, table frames, and text boxes.

Only UIObjects in forms have built-in event methods. You can attach code to those built-in event methods, and a form responds to events. For methods and procedures that work only with forms, use the Form type.

You can also use built-in object variables to refer to UIObjects. This technique is especially useful for creating generalized code.

Many UIObject methods duplicate TCursor methods. The UIObject methods that work with tables work on the underlying table through the visible object. Actions directed to the UIObject that affect the table are immediately visible in the object to which the table is bound. TCursor methods work with a table behind the scenes. Actions that affect the table are not necessarily visible in any object, even if the TCursor is acting on the same table to which a visible object is bound.

Some table operations require Corel Paradox to create temporary tables. Corel Paradox creates these tables in the private directory.

Some table operations are considerably faster with TCursors than with UIObjects. For example, to perform a table-oriented operation that causes a high volume of screen refreshes, you can declare a TCursor, attach it to the object the table is already bound to (e.g., a table frame), perform the operation with the TCursor and resynchronize the display object to the TCursor. When you attach a TCursor to an object bound to a table, the TCursor's record pointer is set to the active record for the object. After you perform a TCursor operation (e.g., a **locate**), the TCursor might point to a different record. To have the object point to the same record as the TCursor, use the **resync** method; to make the TCursor point to the same record as the object, use the **attach** method. For more information, see the example for **insertRecord**.

Methods for the UIObject type

action
atFirst
atLast
attach
bringToFront
broadcastAction
cancelEdit
convertPointWithRespectTo
copyFromArray
copyToArray
copyToToolbar
create
currRecord
delete
deleteRecord
dropGenFilter
edit
empty
end
endEdit
enumFieldNames
enumLocks
enumObjectNames
enumSource
enumSourceToFile
enumUIClasses
enumUIObjectNames

enumUIObjectProperties
execMethod
forceRefresh
getBoundingBox
getGenFilter
getHTMLTemplate
getPosition
getProperty
getPropertyAsString
getRange
getRGB
hasMouse
home
insertAfterRecord
insertBeforeRecord
insertRecord
isContainerValid
isEdit
isEmpty
isLastMouseClickedValid
isLastMouseRightClickedValid
isRecordDeleted
keyChar
keyPhysical
killTimer
locate
locateNext
locateNextPattern
locatePattern
locatePrior
locatePriorPattern
lockRecord
lockStatus
menuAction
methodEdit
methodGet
methodSet
methodDelete
mouseClick
mouseDouble
mouseDown
mouseEnter
mouseExit
mouseMove
mouseRightDouble
mouseRightDown
mouseRightUp
mouseUp
moveTo

moveToRecNo
moveToRecord
nextRecord
nFields
nKeyFields
nRecords
pixelsToTwips
postAction
postRecord
priorRecord
pushButton
recordStatus
resync
rgb
sendToBack
setGenFilter
setPosition
setProperty
setRange
setTimer
skip
switchIndex
twipsToPixels
unDeleteRecord
unlockRecord
view
wasLastClicked
wasLastRightClicked

Print related ObjectPAL methods and examples

action method

Performs a specified action.

Syntax

```
action ( const actionId SmallInt ) Logical
```

Description

action specifies an *actionId* to perform in response to an event. *actionId* is a constant in one of the following action classes:

- ActionDataCommands
- ActionEditCommands
- ActionFieldCommands
- ActionMoveCommands
- ActionSelectCommands

You can also use **action** to send a user-defined action constant to a built-in **action** method. User-defined action constants are simply integers that don't interfere with ObjectPAL's constants. You can use them to signal other parts of an application.

This **action** method is distinct from the built-in **action** method for a form or for any other UIObject. The built-in **action** method for an object responds to an action event, while this method causes an ActionEvent.

Example

{button ,AL(`OPAL_TYPE_UIOBJECT;OPAL_METH_UIMENUACTION;OPAL_METH_AEID;OPAL_METH_AESID;', 0,"Defaultoverview",)} **Related Topics**

action example

The following example is attached to a button's **mouseUp** method. If you click the button, the pointer moves to the next record. If you press and hold SHIFT and click the button, the pointer moves to the next set of records.

The action constants DataFastForward and DataNextRecord behave like the Fast Forward and Next Record buttons on the Toolbar. Assume that *CUSTOMER* refers to a table frame on the form and that *nextRecordOrFast* is a button on the form. Because the *nextRecordOrFast* button is not in the same containership hierarchy as *CUSTOMER*, the action doesn't bubble up to *CUSTOMER* automatically. Instead, the action must be sent to the *CUSTOMER* object explicitly.

```
; nextRecordOrFast::mouseUp
method mouseUp(var eventInfo MouseEvent)
; if the tableFrame isn't active, then move to it
if NOT CUSTOMER.focus then
  CUSTOMER.Name.moveTo()
endIf
; if SHIFT key is down, go to next set of records,
;  otherwise go to next record
if eventInfo.isShiftKeyDown() then
   CUSTOMER.action(DataFastForward)
else
   CUSTOMER.action(DataNextRecord)
endIf
endMethod
```

atFirst method

Reports if the pointer is positioned at the first record of a table.

Syntax

```
atFirst ( ) Logical
```

Description

atFirst returns True if the pointer is positioned at the first record of a table; otherwise, it returns False. **atFirst** respects the limits of restricted views displayed in a linked table frame or multi-record object.

Example

{button ,AL(`OPAL_TYPE_UIOBJECT;OPAL_METH_UIATLA;',0,"Defaultoverview",)} **Related Topics**

atFirst example

The following example assumes that *CUSTOMER* specifies a table frame on the form and that *goToFirstButton* is a button on the form. The code determines the pointer position and moves it to the first record of the *CUSTOMER* table frame.

```
; goToFirstButton::pushButton
method pushButton(var eventInfo Event)
if NOT CUSTOMER.atFirst() then
  CUSTOMER.home()
  ; this has the same effect as:  CUSTOMER.action(DataBegin)
endIf
endMethod
```

atLast method

Reports if the pointer is positioned at the last record in a table.

Syntax

atLast ( ) Logical

Description

atLast returns True if the pointer is positioned at the last record of a table; otherwise, it returns False. **atLast** respects the limits of restricted views displayed in a linked table frame or multi-record object.

Example

{button ,AL(`OPAL_TYPE_UIOBJECT;OPAL_METH_UIATFI;',0,"Defaultoverview",)} **Related Topics**

atLast example

The following example assumes that *CUSTOMER* specifies a table frame on the form and that *goToLastButton* is a button on the form. The code determines the pointer position and moves it to the last record of *CUSTOMER* table frame.

```
; goToLastButton::pushButton
method pushButton(var eventInfo Event)
if NOT CUSTOMER.atLast() then
  CUSTOMER.end()
  ;this has the same effect as:  CUSTOMER.action(DataEnd)
endIf
endMethod
```

attach

Binds a UIObject variable to a specified design object.

Syntax

```
1. attach ( ) Logical
2. attach ( const objectVar UIObject ) Logical
3. attach ( const objectName String ) Logical
4. attach ( const form Form [ , objectName String ] ) Logical
5. attach ( const report Report [ , objectName String ] ) Logical
```

Description

attach binds a UIObject variable to a specified design object. You can also use **attach** to assign a UIObject to an item in an Array.

Syntax 1 binds the variable to the object that called **attach** (*self*).

Syntax 2 binds the variable to another UIObject which is specified by a UIObject variable (*objectVar*) in one of the following ways:

Specification	Example

UIObject variable	`var`
	`  u1, u2 UIObject`
	`endVar`
	`u1.attach()  ; Attach to self.`
	`u2.attach(u1) ; Attach to a UIObject variable.`

UIObject name	`var`
	`  u1 UIObject`
	`endVar`
	`; Attach to an object named nameFld.`
	`u1.attach(nameFld)`

Containership path	`var`
	`  u1  UIObject`
	`  aForm Form`
	`endVar`
	`aForm.open("aform.fsl")`
	`; Attach to an object named aField.`
	`u1.attach(aForm.aPage.aField)`

Syntax 3 binds the variable to another UIObject specified by name in *objectName*. For example, if a form contains a box named *theFrame*, the following statement binds the UIObject variable *ui* to the box:

```
ui.attach("theFrame")
```

Syntax 4 binds the variable to the form which is specified by the Form variable *form*, or to a UIObject in that form which is specified by *objectName*.

Syntax 5 binds the variable to the report which is specified by the Report variable *report*, or to a UIObject which is specified by *objectName*.

Note

- Some of the methods in the UIObject class can be used for forms if you attach a UIObject variable to the form. Syntax 4 of the attach method allows you to attach a UIObject variable to a form so that you can access those methods. For example, to send a mouseUp event to another form's form-level **mouseUp** built-in event method, you must attach a UIObject variable to an open form.

Examples

{button ,AL(`OPAL_TYPE_UIOBJECT;OPAL_METH_UIMOTO;',0,"Defaultoverview",)} Related Topics

attach method examples

Example1 Using the various syntax forms

Example2 Using **attach** to assign a UIObject to an item in an array

attach example 1

The following example displays various forms of the **attach** syntax. First, the code attaches a variable named *objBox* to the active object (*self*) and changes its color. Next, the code attaches *objBox* to another object and uses *objBox* to change that object's color. A second example for of the same syntax opens another form, attaches *objBox* to a box on the second form, and uses *objBox* to change the color of the other form's object.

You can attach to an object name on another form by including the form handle in the object name. Provide the handle to the form in the first argument and the object name on the specified form as a string in the string.

This example assumes the active form contains two boxes, *thisBox* and *thatBox* and the secondary form contains one box, named *otherBox*. The code is attached to *thisBox*.

```
; thisBox::mouseUp
method mouseUp(var eventInfo MouseEvent)
var
  objBox,
  objForm    UIObject
  otherForm  Form
endVar


objBox.attach()           ; binds objBox to thisBox
objBox.color = DarkMagenta


objBox.attach(thatBox)    ; binds objBox to thatBox
objBox.color = Magenta


; assume the form uiattch2.fsl exists and it has
; one object named otherBox
if otherForm.open("uiattch2.fsl") then
  objBox.attach(otherForm.otherBox)
  objBox.color = DarkBlue
  sleep(2000)
  otherForm.close()
endIf


if otherForm.open("uiattch2.fsl") then
  ; notice that the object name is given as a string
  objBox.attach(otherForm, "otherBox")
  objBox.color = LightBlue
  sleep(2000)
  otherForm.close()
endIf


endMethod
```

attach example 2

The following example uses **attach** to assign a UIObject to an item in an array:

```
method pushButton(var eventInfo Event)
   const
      kOneInch = 1440 ; One inch = 1,440 twips.
      kShowHandles = Yes
   endConst


   var
      foForm      Form
      uiTempObj   UIObject
      arObjects   Array[2] UIObject
   endVar


   foForm.create()


   uiTempObj.create(BoxTool, 700, 700, kOneInch, kOneInch, foForm)
   arObjects[1].attach(uiTempObj)


   uiTempObj.create(BoxTool, 700, 2500, kOneInch, kOneInch, foForm)
   arObjects[2].attach(uiTempObj)


   foForm.setSelectedObjects(arObjects, kShowHandles)


endMethod
```

bringToFront method

Displays an object in front of other objects.

Syntax

```
bringToFront ( )
```

Description

bringToFront moves a UIObject to the front drawing layer of a window, displaying it in front of other objects. If UIObject is a form, this method displays the form window in front of all other windows.

This method works in both design and run mode, and you do not have to select the object. The effects of this method might not be noticeable unless the objects partially overlap. This method is also used if you want to rearrange the objects' tab order.

Note

- When you change the front-to-back positions of objects, you also change their tab order. Objects always tab from back to front.

Example

{button ,AL(`OPAL_TYPE_UIOBJECT;OPAL_METH_UISENDTOBACK;',0,"Defaultoverview",)} Related Topics

bringToFront example

In the following example, the **pushButton** method for a button displays an animated sequence of twelve bitmaps. This code uses two **for** loops and the **bringToFront** method to cycle through the bitmaps forward and backward.

```
;btn1 :: pushButton
method pushButton(var eventInfo Event)
   var
      siCounter SmallInt
   endVar


   ;Cycle through bitmaps.
   for siCounter from 1 to 12
      ; Assume the bitmap objects have names like bmp1, bmp2, etc.
      pge1.("bmp" + string(siCounter)).bringToFront()
      sleep(100)
   endFor


   ;Cycle through bitmaps in reverse order.
   for siCounter from 11 to 1 step -1
      pge1.("bmp" + string(siCounter)).bringToFront()
      sleep(100)
   endFor
endMethod
```

broadcastAction method

Broadcasts an action to an object and the objects it contains.

Syntax

```
broadcastAction (const actionID SmallInt)
```

Description

broadcastAction sends the ActionEvent specified in *actionID* to an object, and then sequentially to each object it contains. The action is sent depth-first through the containership hierarchy, not breadth-first. By default, contained objects bubble the action up through the hierarchy.

For example, suppose a page named *thePage* contains two boxes (*boxOne* and *boxTwo*) and *boxOne* contains a button *btnOne*. A call to **thePage.broadcastAction(actionID)** sends the action specified by *actionID* to the objects in the following order:

1. *thePage* (specified by dot notation)
2. *boxOne* (contained by thePage)
3. *btnOne* (contained by boxOne, at a lower level in the hierarchy)
4. *boxTwo* (also contained by thePage, at the same level as boxOne in the hierarchy)

The value of *actionID* can be a user-defined action constant or a constant from one of the following Action classes:

- ActionDataCommands
- ActionEditCommands
- ActionFieldCommands
- ActionMoveCommands
- ActionSelectCommands

Example

{button ,AL(`OPAL_TYPE_UIOBJECT;OPAL_METH_UIACTI;OPAL_METH_UIMENUACTION;OPAL_METH_UIPOSTACTION;OPAL_TYPE_ACTIONEVENT;OPAL_ACTIONEVENT_USERDEFINEDCONSTANTS;',0,"Defaultoverview",)} **Related Topics**

broadcastAction example

In the following example, the form's built-in **action** method uses **broadcastAction** to send all the objects in the page *pge1* a user-defined action. When the form switches to edit mode, it sends *UserAction + 1*. When the form leaves edit mode, it sends *UserAction + 2*. Each field's label then uses the user-defined action to color each label Red or Black.

The following code is attached to the form's built-in **action** method:

```
;frm1 :: action
method action(var eventInfo ActionEvent)


   if eventInfo.isPreFilter() then
      ;// This code executes for each object on the form:

   else
      ;// This code executes only for the form:

      switch
         case eventInfo.id() = DataBeginEdit :
               pge1.broadcastAction(UserAction + 1)


         case eventInfo.id() = DataEndEdit :
               pge1.broadcastAction(UserAction + 2)
      endSwitch


   endIf


endmethod
```

The following code is attached to each label's built-in **action** method:

```
;label :: action
method action(var eventInfo ActionEvent)
   ;Duplicate this code on each object (or create a prototype object)
   ;you wish toggle the font color from black to red when
   ;the form goes in and out of edit mode.


switch
   case eventInfo.id() = UserAction + 1 : self.font.color = Red
   case eventInfo.id() = UserAction + 2 : self.font.color = Black
endSwitch
endmethod
```

cancelEdit method

Cancels record changes without ending Edit mode.

Syntax

cancelEdit ( ) Logical

Description

cancelEdit cancels changes you've made to the active record. This method returns True if successful; otherwise, it returns False. To cancel record changes, use **cancelEdit** before moving the pointer from the active record. If you move the pointer, changes to the record are committed.

cancelEdit has the same effect as the action constant DataCancelEdit. This means that the following statements are equivalent:

```
obj.cancelEdit()
obj.action(DataCancelEdit)
```

Example

{button ,AL(`OPAL_TYPE_UIOBJECT;OPAL_METH_UICURR;OPAL_METH_UIEDIT;OPAL_METH_UIECED;',0,"Defaultoverview",)} Related Topics

cancelEdit example

The following example attaches a UIObject variable (*noChange*) to a table frame (*CUSTOMER*). (From then on *noChange* is used as a handle to the table frame.) The code searches for a value in the *Customer* table, and, if found, changes the value. Before leaving the record, the change is canceled using the **cancelEdit** method. This example assumes that you have one page on the form (*pageOne*), a table frame attached to the *Customer* table, and a button (*CancelEditButton)*.

```
; CancelEditButton::pushButton
method pushButton(var eventInfo Event)
var
  noChange UIObject
endVar


noChange.attach()
noChange.attach(pageOne.CUSTOMER)
noChange.edit()
if noChange.locate("Name", "Unisco") then
  noChange."Name" = "Jones"   ; prepare to change the record
  msgInfo("noChange.'Name'", noChange."Name".value)
  noChange.cancelEdit()        ; belay that order!
                           ; record not changed,
endIf
noChange.endEdit()         ; exit Edit mode


endMethod
```

convertPointWithRespectTo method

Changes the frame of reference for calculating the coordinates of a point.

Syntax

```
convertPointWithRespectTo ( const otherUIObject UIObject, const oldPoint Point, var
convertedPoint Point )
```

Description

convertPointWithRespectTo changes the frame of reference for calculating the coordinates of a point. Coordinates are usually calculated relative to the upper-left corner of the object's container (or the container's frame, in the case of an ellipse). This method instead calculates a point's position relative to the upper-left corner of the object specified in *otherUIObject*.

Example

{button ,AL(`OPAL_TYPE_UIOBJECT;OPAL_TYPE_POINT;',0,"Defaultoverview",)} Related Topics

convertPointWithRespectTo example

The following example retrieves and displays the position of an object named *innerBox*. *innerBox* is contained by *outerBox* on a page named *pageOne*. First, the position of *outerBox* relative to the upper-left corner of the page is calculated. Next, the position of *innerBox*, relative to the upper-left corner of *outerBox* is calculated. Finally, the position of *innerBox* is converted with respect to the page, allowing you to determine the distance between *innerBox* and the top and left edges of the page.

```
; alignInnerBox::pushButton
method pushButton(var eventInfo Event)
var
  innerPos,
  outerPos,
  convertedPos  Point
  x, y, w, h    LongInt
endVar


outerBox.getPosition(x, y, w, h)
outerPos = point(x, y) ; convert x and y from
outerPos.view("Outer box position") ; outerBox to a point
innerBox.getPosition(x, y, w, h)
innerPos = point(x, y)
innerPos.view("Inner box position unconverted")
; how far is innerPos from the upper left corner of the page?
outerBox.convertPointWithRespectTo(pageOne, innerPos, convertedPos)
convertedPos.view("Inner box position converted")
endMethod
```

copyFromArray method

Copies data from an array to a table record.

Syntax

```
copyFromArray ( const ar Array[ ] AnyType) Logical
```

Description

copyFromArray copies data from an array *ar* to a UIObject (usually a table frame or multi-record object). The first element of the array is copied to the first field of the table, the second element to the second field, and so on until the array is exhausted or the record is full.

This method fails if you copy an unassigned array element or if the structures do not match. (This can never happen if the array was created by **copyToArray**, which assigns a blank value if a field is blank.) This method also fails if the form is not in Edit mode. If there are more elements in the array than fields in the record, the extra elements are ignored.

Example

{button ,AL(`OPAL_TYPE_UIOBJECT;OPAL_METH_UICTO;',0,"Defaultoverview",)} Related Topics

copyFromArray example

The following example assumes that a form contains a table frame named *CUSTNAME*. The *CUSTNAME* table has three fields: Last name, A20; First name, A20; and Middle Initial, A1. This code edits *CUSTNAME*, creates an array with three elements, creates a new record in *CUSTNAME* and copies data from the array to the record.

```
; createRecord::pushButton
method pushButton(var eventInfo Event)
var
  nameArray Array[3] String
endvar
CUSTNAME.edit()            ; start Edit mode
nameArray[1] = "Hall"      ; fill the array with the record to insert
nameArray[2] = "Robert"
nameArray[3] = "A"
CUSTNAME.action(DataInsertRecord) ; insert a blank record first
CUSTNAME.copyFromArray(nameArray) ; copy the array to the new record
CUSTNAME.endEdit()
endMethod
```

copyToArray method

Copies data from a record to an array.

Syntax

```
copyToArray ( var ar Array [ ] AnyType ) Logical
```

Description

copyToArray copies fields from the record of a UIObject (usually a table frame or multi-record object) to an array. You must declare the array as AnyType, or as a type that matches each field in the table. If the array is resizeable, it expands to hold the number of fields in the record. If the array is not resizeable, it discards the fields it cannot hold.

The value of the first field is copied to the first element of the array, the value of the second field to the second element, and so on. The size of the array is equal to the number of fields in the record. The record number field and any display-only or calculated fields are not copied to the array.

Example

{button ,AL(`OPAL_TYPE_UIOBJECT;OPAL_METH_UICFROM;',0,"Defaultoverview",)} Related Topics

copyToArray example

The following example assumes that there are two table frames on a form, named *CUSTOMER* and *CUSTARC*, and one button, named *archiveButton*. The form itself is renamed *thisForm*. When *archiveButton* is pushed, the active record in *CUSTOMER* is moved to *CUSTARC*.

This code looks at the Editing property of the form; if it's False, this code starts Edit mode. **copyToArray** then copies the active record in *CUSTOMER* to the *arcRecord* array and deletes the active record. If the active record can't be locked and deleted, it is not copied to the target table *CUSTARC*. If the record can be deleted, **copyToArray** writes the contents of the array to a new blank record in the target table.

```
; archiveButton::pushButton
method pushButton(var eventInfo Event)
var
  arcRecord Array[] String
endVar


; check to see if form is in edit mode
if thisForm.Editing = False then ; if not, then start
  CUSTOMER.action(DataBeginEdit)
endIf


; move the active record from CUSTOMER to archive in CUSTARC
CUSTOMER.copyToArray(arcRecord)
arcRecord.view()           ; take a look at the array
; if the record can't be locked, it won't be deleted
if CUSTOMER.deleteRecord() = True then
  ; if it is deleted, then copy it to the archive table
  CUSTARC.insertRecord()            ; insert blank record
  CUSTARC.copyFromArray(arcRecord)  ; copy array to blank record
endIf


endMethod
```

copyToToolbar method

Copies an object to the Toolbar where it can be used as a prototype object.

Syntax

copyToToolbar ( ) Logical

Description

copyToToolbar copies an object (including its properties and methods) to the Toolbar. New objects created using the corresponding Toolbar tool will have the new properties, and existing objects do not change.

For example, create a box (interactively or using ObjectPAL) and set its color to red, and add code to its built-in **mouseClick** method. If you copy this box to the Toolbar, all new boxes you create are red and have the same code attached to the **mouseClick** method.

copyToToolbar copies all component objects in a compound object. For example, when you copy a labeled field object, you copy the field object, the label, and the edit region. Tables include headers, labels, records, and fields. Multi-record objects include records only. Crosstabs include cells and fields. They can distinguish the three different cell types, so you can have three different types of fields which have different colors, and so on.

You can also use **copyToToolbar** to copy the component objects separately. However, if an object contains objects, but is not a compound object, the contained objects are not copied.

Changes you make using **copyToToolbar** apply only to the current Corel Paradox session. To save the tool's new properties to the next session, call **saveStyleSheet**.

If an object does not have a corresponding tool on the Toolbar, Corel Paradox copies its properties and methods to a hidden tool. All new objects of that type will have those properties and methods. For example, the Toolbar does not have a tool for creating a page. However, you can set a page's properties and methods, and call **copyToToolbar** to apply the same properties and methods to a new page.

Example

{button ,AL(`OPAL_TYPE_UIOBJECT;OPAL_METH_FOGETPROTOPROPERTY;OPAL_METH_FOSAVESTYLESHEET;OPAL_METH_FOSELECTCURRENTTOOL;OPAL_METH_FOSETPROTOPROPERTY;OPAL_METH_SYGETDEFAULTPRINTERSTYLESHEET;OPAL_METH_SYGETDEFAULTSCREENSTYLESHEET;OPAL_METH_SYSETDEFAULTPRINTERSTYLESHEET;OPAL_METH_SYSETDEFAULTSCREENSTYLESHEET;',0,"Defaultoverview",)} **Related Topics**

copyToToolbar example

In the following example, a button named *btnCreateStyleSheet* uses **enumObjectNames** to fill an array *arObjNames*. A **for** loop cycles through the array and copies the object to the Toolbar using **copyToToolbar**. A call to **saveStyleSheet** creates (or overwrites) a style sheet with the name in the String variable *stSheet*. You can paste the following code into the **pushButton** method for a button on any form you want to use as a style sheet:

```
;btnCreateStyleSheet :: pushButton
method pushButton(var eventInfo Event)
   var
      f                  Form
      stSheet      String
      arObjNames   Array[] Anytype
      siCounter   SmallInt
   endVar


   f.attach()               ; Attach to this form.
   f.enumObjectNames(arObjNames)   ; Fill array with object names.


   ; Prompt user for name of new style sheet.
   stSheet = "Style sheet name"
   stSheet.view("Enter name of style sheet")


   if stSheet = "Style sheet name" then   ; If variable was not changed,
      return                  ; quit the operation.
   endIf


   for siCounter from 1 to arObjNames.size()   ; Cycle through array
      copyToToolbar(f.(arObjNames[siCounter]))   ; and copy objects to
   endFor                         ; the Toolbar.


   if not f.saveStyleSheet(stSheet, True) then
      errorShow("Error saving style sheet", "Check path & filename")
   endIf
endMethod
```

create method

Creates an object.

Syntax

```
1. create ( const objectType SmallInt, const x LongInt, const y LongInt, const w LongInt, const
h LongInt [, const container UIObject ] )

2. create ( const nativeObject Binary, const container UIObject ) Logical
```

Description

create creates the object specified in **objectType** (use one of the UIObjectTypes constants) at a position specified in **x** and **y**, with a width specified in **w**, and a height specified in **h. x**, **y**, **w**, and **h** are assumed to be in twips. The optional argument container specifies a container object for the new object.

Syntax 2, uses **create** to create the object specified by *nativeObject*. *nativeObject* is a binary object that can be generated by pasting a UIObject (Corel Form Object) from the Clipboard. **create** works only in form design mode. **create** returns True if successful; otherwise it returns False.

Note

- When you use **create** to create an object, the object is invisible. To make it visible, set its Visible property to True. To delete an objet at run time use the **delete** method.

Example

{button ,AL(`OPAL_TYPE_UIOBJECT;OPAL_METH_UIDELETE;OPAL_METH_UIMETHODSET;',0,"Defaultoverview",)} Related Topics

create example

In the following example, code is attached to the **mouseUp** method for *pageOne* on a form. This example creates a box, names it *Fred*, colors it blue, and makes it visible. The code then creates an ellipse whose size position is specified in *Fred*, and whose container is set to *Fred*.

```
; pageOne::mouseUp
method mouseUp(var eventInfo MouseEvent)
const

    kOneInch = 1440 ; One inch = 1,440 twips.

endConst

var
  ui UIObject

  fm Form
endvar


; create a Blue box, named Fred, and make it visible
ui.create(BoxTool, 144, 144, 2 * kOneInch, 2 * kOneInch)
ui.Name = "Fred"
ui.Color = Blue
ui.Visible = True
; create a Green ellipse inside Fred, named Bill
fm.attach()

ui.create (EllipseTool, 288, 288, kOneInch, kOneInch, fm.Fred)
ui.Name = "Bill"
ui.Color = Green
ui.Visible = True
endMethod
```

currRecord method

Reads the active record into the record buffer.

Syntax

currRecord ( ) Logical

Description

currRecord cancels changes you've made to the active record, and displays a refreshed version from saved data. **currRecord** leaves a locks on locked records. This method returns True if successful; otherwise, it returns False.

currRecord has the same effect as the action constant DataRefresh. This means that the following statements are equivalent:

```
obj.currRecord()
obj.action(DataRefresh)
```

Example

{button ,AL(`OPAL_TYPE_UIOBJECT;OPAL_METH_UICANCELEDIT;',0,"Defaultoverview",)} **Related Topics**

currRecord example

The following examples assumes that a form contains a table frame bound to *Orders*;

```
;refreshRecord::pushButton
method pushButton(var eventInfo Event)
ORDERS.edit()            ; start edit
ORDERS.Amount_Paid = 321.45   ; make a change
message("Watch closely now.")
sleep(2000)
ORDERS.currRecord()      ; refreshes record from disk,
                         ; any changes are lost, record
                         ; is not locked
if ORDERS.recordStatus("Locked") then
  msgInfo("FYI", "The record is still locked.")
endIf
endMethod
```

delete method

Deletes an object from a form.

Syntax

```
delete ( )
```

Description

delete deletes an object from a form at run time.

Example

{button ,AL(`OPAL_TYPE_UIOBJECT;OPAL_METH_UICREATE;OPAL_METH_UIMETHODSET;',0,"Defaultoverview",)} Related Topics

delete example

The following examples assumes that a form contains a method that creates a box named *Fred* and an ellipse inside *Fred* named *Bill*. Because these objects are created at run time, they can't be referenced directly by this method. This code attaches to the object using a string evaluated at run time. See the example for **create** for details about the **mouseUp** method (on the form) that creates the objects to be deleted.

```
; pageOne::mouseRightUp
method mouseRightUp(var eventInfo MouseEvent)
var
  ui   UIObject
endVar


;  Fred and Bill are objects created by the mouseUp method
;  for pageOne of this form. Because they are created at
;  run time, you can't directly refer to them as objects in
;  code. Consequently, attach is used to attach the ui var
;  to the string "Fred.Bill", which is evaluated at run time.
;  As long as mouseUp is called before mouseRightUp, those
;  objects will exist.
if ui.attach("Fred.Bill") then
  ui.delete()
  ui.attach("Fred")
  ui.delete()
  {This would do the same thing as previous four lines,
    because Fred contains Bill at run time:
    ui.attach("Fred")
    ui.delete()
  }
endIf
endMethod
```

deleteRecord method

Deletes the active record from a table.

Syntax

deleteRecord ( ) Logical

Description

deleteRecord deletes the active record from a table without prompting for confirmation. This method returns True if successful; otherwise, it returns False. Deleted dBASE tables can be restored after **deleteRecord**, but Corel Paradox tables cannot.

deleteRecord has the same effect as the action constant DataDeleteRecord. This means that the following statements are equivalent:

```
obj.deleteRecord()
obj.action(DataDeleteRecord)
```

Example

{button ,AL(`OPAL_TYPE_UIOBJECT;OPAL_METH_UIEMP;OPAL_METH_UIIREC;OPAL_METH_UIIAFT;OPAL_METH_UIIBEF;',0,"Defaultoverview",)} Related Topics

deleteRecord example

The following example assumes that there are two table frames on a form, *CUSTOMER* and *CUSTARC*, and one button, named *archiveButton*. The form is renamed *thisForm*. When *archiveButton* is pushed, the active record in *CUSTOMER* is moved to *CUSTARC*.

To begin, this method determines the Editing property of the form; if it is False, this code starts Edit mode. This code then copies the active record in *CUSTOMER* to the *arcRecord* array and deletes the active record. If the active record can't be locked and deleted, it is not copied to the target table. If the record can be deleted, this code writes the contents of the array to the target table in a new blank record.

```
; archiveButton::pushButton
method pushButton(var eventInfo Event)
var
  arcRecord Array[] String
endVar


; check to see if form is in edit mode
if thisForm.editing = False then ; if not, then start
  CUSTOMER.action(DataBeginEdit)
endIf


; move the active record from CUSTOMER to archive in CUSTARC
CUSTOMER.copyToArray(arcRecord)
arcRecord.view()           ; take a look at the array
; if the record can't be locked, it won't be deleted
if CUSTOMER.deleteRecord() = True then
  ; if it is deleted, then copy it to the archive table
  CUSTARC.insertRecord()
  CUSTARC.copyFromArray(arcRecord)
endIf


endMethod
```

dropGenFilter method

Removes the filter criteria associated with a field, multi-record object, or table frame.

Syntax

dropGenFilter ( ) Logical

Description

dropGenFilter removes the filter criteria associated with a UIObject. All associated indexes and ranges remain.

Example

{button ,AL(`OPAL_TYPE_UIOBJECT;OPAL_METH_UIGETGENFILTER;OPAL_METH_UISETGENFILTER;OPAL_METH_UISETRANGE;OPAL_METH_TCDROPGENFILTER;',0,"Defaultoverview",)} **Related Topics**

dropGenFilter example

In the following example, a form's data model contains the *Orders* and *Lineitem* tables linked 1:M. The form also contains a button named *btnDropFilters*. The **pushButton** method for *btnDropFilters* uses **dropGenFilter** on one UIObject connected to each table in the data model. This code allows you to remove filter criteria from complex forms.

```
;btnDropFilters :: pushButton
method pushButton(var eventInfo Event)
   ; Order_No is a field object bound to
   ; the Order No field in the Orders table.
   Order_No.dropGenFilter()


   ; LINEITEM is a table frame bound to the Lineitem table.
   LINEITEM.dropGenFilter()
endMethod
```

edit method

Puts a table in Edit mode.

Syntax

```
edit ( ) Logical
```

Description

edit puts all tables on a form in Edit mode, allowing you to make changes. If the form is already in Edit mode, **edit** is ignored.

In Edit mode, record changes are posted when the focus moves off the record, when the table receives a DataPostRecord or DataUnlockRecord action, or when **endEdit** is executed. Use **cancelEdit** to cancel changes to the record before moving on.

edit has the same effect as the action constant DataBeginEdit. This means that the following statements are equivalent:

```
obj.edit()
obj.action(DataBeginEdit)
```

Example

{button ,AL(`OPAL_TYPE_UIOBJECT;OPAL_METH_UICANCELEDIT;OPAL_METH_UIECED;',0,"Defaultoverview",)} Related Topics

edit example

The following examples assumes that a form contains a table frame bound to the *Orders* table, and one button, named *changeDate*. The **pushButton** method for *changeDate* examines the Sale Date and Ship Date fields of the active record, and updates Sale Date if Ship Date is less than Sale Date. Once the transaction is complete, **endEdit** posts the record and ends Edit mode.

```
; changeDate::pushButton
method pushButton(var eventInfo Event)


; first, see if you want to change Ship Date
if ORDERS."Sale Date".value > ORDERS."Ship Date".value then
  ; start Edit mode for the form
  ORDERS.edit()
  ; if Sale Date is later than Ship Date, change Ship Date
  ORDERS."Ship Date".value = ORDERS."Sale Date".value + 5
  ORDERS.endEdit()      ; end editing changes to the record
                        ; can't be canceled
endIf


endMethod
```

empty method

Deletes all records from a table.

Syntax

empty ( ) Logical

Description

empty deletes all records from a table without prompting for confirmation. The table does not have to be in Edit mode, but a write lock (at least) is required. This operation cannot be undone for Corel Paradox tables, and does not affect SQL tables.

empty removes information from the table, without deleting the table itself. Compare this method to **delete** (Table type), which does delete the table.

empty tries to place a write lock on the table. **empty** must delete each record one at a time. For dBASE tables, this method flags all records as deleted, without removing them from the table. Records can be undeleted from a dBASE table using the **unDeleteRecord** method (unless they have been removed using the **compact** (TCursor type) method).

 **Example**

{button ,AL(`OPAL_TYPE_UIOBJECT;OPAL_METH_UIDELE;OPAL_METH_TBDELETE;OPAL_METH_TBEMPT;OPAL_METH_TCEMPT;',0,"Defaultoverview",)} Related Topics

empty example

The following example assumes that a form has three buttons: *createTable*, *emptyTable*, and *deleteTable*. *createTable* creates a copy of the *Orders* table named *TmpOrder* and places a table frame on the form, and binds *TmpOrder* to it. *emptyTable* deletes all the records from *TmpOrder*. *deleteTable* removes the table frame, removes the table from the form's data model, and deletes the temporary table.

The following code attaches to the *createTable* button:

```
; createTable::pushButton
method pushButton(var eventInfo Event)
var
  tbl Table
  ui  UIObject
endVar


tbl.attach("Orders.db")
tbl.copy("TmpOrder.db")  ; Copy Orders to TmpOrder.


ui.create(TableFrameTool, 720, 720, 4320, 1440) ; Create a TableFrame.
ui.TableName = "TmpOrder.db" ; This also adds table to data model.
ui.Visible = True


endMethod
```

The following code attaches to the *emptyTable* button:

```
; emptyTable::pushButton
method pushButton(var eventInfo Event)
var
  ui  UIObject
endVar


if ui.attach("TMPORDER") then
  if msgYesNoCancel("Empty",
    "Delete all records from this table?") = "Yes" then
    ui.empty()  ; Deletes all records from the TMPORDERS table.


  endIf
endIf


endMethod
```

The following code attaches to the *deleteTable* button:

```
; deleteTable::pushButton
method pushButton(var eventInfo Event)
var
  tbl Table
  ui  UIObject
endVar


; Clean up.
if ui.attach("TMPORDER") then
  ui.delete()                   ; Delete table frame.
  DMRemoveTable("TmpOrder.db")  ; Remove table from data model.
  tbl.attach("TmpOrder.db")
  tbl.delete()                  ; Delete table.
endIf
endMethod
```

end method

Moves to the last record in a table.

Syntax

end ( ) Logical

Description

end moves to the last record in a table.

end has the same effect as the action constant DataEnd. This means that the following statements are equivalent:

```
obj.end()
obj.action(DataEnd)
```

Example

{button ,AL(`OPAL_TYPE_UIOBJECT;OPAL_METH_UIHOME;OPAL_METH_UINEXTR;OPAL_METH_UIPRIO;OPAL_METH_UICURR;OPAL_METH_UISKIP;OPAL_METH_UIMOTO;',0,"Defaultoverview",)} Related Topics

end example

The following example moves to the last record in the *Customer* table. Assume that *Customer* is bound to a table frame on the form, and that *moveToEnd* is a button on the form.

```
; moveToEnd::pushButton
method pushButton(var eventInfo Event)
CUSTOMER.end()  ; move to the last record
                ; same as:  CUSTOMER.action(DataEnd)
msgInfo("At the last record?", CUSTOMER.atLast())
endMethod
```

endEdit method

Removes a table from Edit mode and posts changes to the active record

Syntax

endEdit ( ) Logical

Description

endEdit removes a table from Edit mode and posts changes to the active record.

endEdit has the same effect as the action constant DataEndEdit. This means that the following statements are equivalent:

```
obj.endEdit()
obj.action(DataEndEdit)
```

Example

{button ,AL(`OPAL_TYPE_UIOBJECT;OPAL_METH_UICANCELEDIT;OPAL_METH_UIEDIT;',0,"Defaultoverview",)} Related Topics

endEdit example

See the **edit** example.

enumFieldNames method

Fills an array with the names of fields in a table.

Syntax

```
enumFieldNames ( var fieldArray Array[ ] String ) Logical
```

Description

enumFieldNames fills *fieldArray* with the names of the fields in a table. *fieldArray* is a resizeable array that you must declare and pass as an argument. If *fieldArray* already exists, this method overwrites it without asking for confirmation. **enumFieldNames** returns True if it succeeds; otherwise, it returns False.

Example

{button ,AL(`OPAL_TYPE_UIOBJECT;OPAL_METH_UIEONA;',0,"Defaultoverview",)} Related Topics

enumFieldNames example

The following example uses **enumFieldNames** to write the field names from the *Orders* table to an array named *fieldNames*. Assume that a form has a table frame bound to *Orders* and a button named *getFieldNames*.

```
; getFieldNames::pushButton
method pushButton(var eventInfo Event)
var
  fieldNames Array[] String
endVar
ORDERS.enumFieldNames(fieldNames)
fieldNames.view()
endMethod
```

enumLocks method

Creates a Corel Paradox table listing the locks currently applied to a UIObject, and returns the number of locks.

Syntax

```
enumLocks ( const tableName String ) LongInt
```

Description

enumLocks creates the Corel Paradox table specified in *tableName*. *tableName* lists the locks currently applied to the table object. If *tableName* exists, this method overwrites it without asking for confirmation. If *tableName* is open, **enumLocks** fails. For dBASE tables, this method lists only the lock that you've placed (not all locks currently on the table).

You can specify an alias or path in *tableName*. If an alias or path is not specified, Corel Paradox creates *tableName* in the working directory.

The following table displays the structure of *tableName*:

Field name	Field type	Description
UserName	A15	User name
LockType	A32	Lock type (e.g., Table Write Lock)
NetSession	N	Net level session number
Session	N	BDE session number (for locks placed by BDE)
RecordNumber	N	Record number (for record locks or image locks; otherwise, 0)

Example

enumLocks example

In the following example, the built-in **pushButton** method for the *showLocks* button creates a table listing the locks specified for the *Customer* table:

```
; showLocks::pushButton
method pushButton(var eventInfo Event)
var
  obj       UIObject
  howMany   LongInt
  enumTable TableView
endVar
obj.attach(CUSTOMER)            ; table frame on form
lock("Customer", "Write")       ; put a write lock on Customer
howMany = obj.enumLocks("lockenum.db")  ; enumerate locks
message("There are ", howMany, " locks on Customer table.")
enumTable.open("lockenum.db")   ; show the resulting table
enumTable.wait()
enumTable.close()
endMethod
```

enumObjectNames method/procedure

Fills an array with the names of the objects in a form.

Syntax

```
enumObjectNames ( var objectNames Array[ ] String )
```

Description

enumObjectNames fills an array with object names. *arrayName* is a resizeable array that you declare and pass as an argument. If *arrayName* already exists, this method overwrites it without asking for confirmation.

enumObjectNames returns the names of bound and unbound objects, beginning with the object that called the method, including the paths to objects that object contains. To enumerate all objects in a form, start **enumObjectNames** with the form. To enumerate all objects in a page, start it with the page. To enumerate all objects in a box, start it with the box.

To list object names in a table use **enumUIObjectNames**.

Example

{button ,AL(`OPAL_TYPE_UIOBJECT;OPAL_METH_UIEONT;OPAL_METH_UIEOPT;OPAL_METH_UIECTT;',0,"Defaultoverview",)} Related Topics

enumObjectNames example

The following example demonstrates the difference between **enumObjectNames** (which lists object names in an array) and **enumUIObjectNames** (which lists object names in a table). In this example, the **pushButton** method for *getObjectNames* writes all object names on the form to an array and to a table.

```
; getObjectNames::pushButton
method pushButton(var eventInfo Event)
   var
      foThisForm   Form
      arObjNames   Array[] String
      stTbName   String
      tvObjNames   TableView
   endVar


   stTbName = "objTable.db"
   foThisForm.attach() ; Get a handle to the current form.


   foThisForm.enumObjectNames(arObjNames)
   arObjNames.view("Objects in this form:")


   foThisForm.enumUIObjectNames(stTbName)
   tvObjNames.open(stTbName)
endMethod
```

enumSource method

Fills a table with the source code of the methods on a form.

Syntax

```
enumSource ( const tableName String, [ const recurse Logical ] ) Logical
```

Description

enumSource fills a table with the source code of the methods on a form. If *tableName* already exists, this method overwrites it without asking for confirmation. You can specify an alias or path in *tableName*. If an alias or path is not specified, Corel Paradox creates *tableName* in the working directory.

The following table displays the structure of the table created by **enumSource**:

Field name	Type	Description
Object	A128	Object name
MethodName	A128	Method name
Source	M64	ObjectPAL code

If *recurse* is False, **enumSource** returns the method definitions for overridden methods on the active object. To include the source code for overridden methods on objects contained by the active object, set *recurse* to True.

If *recurse* is True, **enumSource** returns the definitions for overridden methods, beginning with the object that called this method, and including paths to objects that object contains. To enumerate all objects in a form, start **enumSource** with the form. To enumerate all objects in a page, start it with the page. To enumerate all objects in a box, start it with the box.

Example

{button ,AL(`OPAL_TYPE_UIOBJECT;OPAL_METH_UIENUMSOURCETOFILE;OPAL_METH_UIEOPT;OPAL_METH_UIECTT;OPAL_METH_FOENUMSOURCE;',0,"Defaultoverview",)} Related Topics

enumSource example

The following example uses **enumSource** to retrieve the source code for the entire form and to retrieve only the source code for a button named *btnCancel*:

```
; getObjectNames::pushButton
const
   kRecurse   = Yes
   kNoRecurse = No
endConst


method pushButton(var eventInfo Event)
   var
      foThisForm   Form
      stTbName   String
      tvSource   TableView
   endVar

   stTbName = "objSrc.db"
   foThisForm.attach() ; Get a handle to the current form.

   foThisForm.enumSource(stTbName, kRecurse)
   tvSource.open(stTbName)

   ; Suspend execution until you close the table view.
   tvSource.wait()

   btnCancel.enumSource(stTbName, kNoRecurse)
   tvSource.open(stTbName)
endMethod
```

enumSourceToFile method

Writes the source code for a form or an object to a text file.

Syntax

enumSourceToFile (const ***fileName*** String [**,** const ***recurse*** Logical] **)** Logical

Description

enumSourceToFile writes the source code of the methods on a form to a text file. If *fileName* already exists, this method overwrites it without asking for confirmation. You can specify an alias or path in *fileName* . If an alias or path is not specified, Corel Paradox creates *fileName* in the working directory.

If *recurse* is False, **enumSourceToFile** returns the method definitions for overridden methods on the active object. To include the source code for overridden methods on objects contained by the active object, set *recurse* to True.

If *recurse* is True, **enumSourceToFile** returns the definitions for overridden methods, beginning with the object that called this method, and including paths to objects that object contains. To enumerate all objects in a form, start **enumSourceToFile** with the form. To enumerate all objects in a page, start it with the page. To enumerate all objects in a box, start it with the box.

Example

{button ,AL(`OPAL_TYPE_UIOBJECT;OPAL_METH_UIENUMSOURCE;OPAL_METH_UIEOPT;OPAL_METH_UIECTT;OPAL_METH_UIENUMSOURCETOFILE;',0,"Defaultoverview",)} **Related Topics**

enumSourceToFile example

The following example uses **enumSourceToFile** to retrieve the source code for the entire form and to retrieve only the source code for a button named *btnCancel*:

```
; getObjectNames::pushButton
const
   kRecurse   = Yes
   kNoRecurse = No
endConst


method pushButton(var eventInfo Event)
   var
      foThisForm   Form
   endVar


   foThisForm.attach() ; Get a handle to the current form.
   foThisForm.enumSource("formSrc.txt", kRecurse)


   btnCancel.enumSource("btnSrc.txt", kNoRecurse)
endMethod
```

enumUIClasses procedure

Writes a list of UIObject classes to a table.

Syntax

```
enumUIClasses ( const tableName String ) Logical
```

Description

enumUIClasses creates a table named *tableName* that contains a list of all UIObject classes (e.g., bitmap, box, and field) and the names of their associated properties. You can specify an alias or path in *tableName*; if an alias or path is not specified, Corel Paradox creates *tableName* in the working directory.

The following table displays the structure of the table created by **enumUIClasses**:

Field name	Type	Description
ClassName	A32	Name of object class
PropertyName	A64	Name of property

Example

{button ,AL(`OPAL_TYPE_UIOBJECT;OPAL_METH_UIEONT;OPAL_METH_UIEOPT;',0,"Defaultoverview",)}

Related Topics

enumUIClasses example

The following example writes a list of UIObject classes, including their types and properties, to a table named *Tmpclass*:

```
; writeClasses::pushButton
method pushButton(var eventInfo Event)
enumUIClasses("TmpClass.db")
endMethod
```

enumUIObjectNames method/procedure

Writes the names of each object in a form to a table.

Syntax

enumUIObjectNames (const *tableName* String) Logical

Description

enumUIObjectNames writes the names of each object in a form to a table. You can specify an alias or path in *tableName*. If an alias or path is not specified, Corel Paradox creates *tableName* in the working directory.

The following table displays the structure of the table created by **enumUIObjectNames** :

Field name	Type	Description
ObjectName	A128	Name of object
ObjectClass	A32	Type of object (e.g., button)

enumUIObjectNames returns the names of bound objects and unbound objects, beginning with the object that called this method, and including paths to any objects that object contains. To enumerate all objects in a form, make **enumUIObjectNames** start with the form. To enumerate all objects in a page, make it start with the page. To enumerate all objects in a box, make it start with the box.

To write the form's object names to an array, use **enumObjectNames**.

Example

{button ,AL(`OPAL_TYPE_UIOBJECT;OPAL_METH_UIEONA;OPAL_METH_UIEOPT;OPAL_METH_UIECTT;OPAL_METH_FOENNAM;',0,"Defaultoverview",)} Related Topics

enumUIObjectNames example

See the **enumObjectNames** example.

enumUIObjectProperties method/procedure

Lists the properties of an object.

Syntax

```
1. enumUIObjectProperties ( const tableName String ) Logical
2. enumUIObjectProperties ( const properties DynArray[ ] String ) Logical
```

Description

enumUIObjectProperties lists the properties of an object in a table or a dynamic array (DynArray).

Syntax 1 writes the data to the Corel Paradox table specified in *tableName*. You can specify an alias or path in *tableName*. If an alias or path is not specified, Corel Paradox creates *tableName* in the working directory. If the table already exists, Corel Paradox overwrites it without asking for confirmation.

The table lists the properties of the specified object, and the properties of the objects it contains. The following table displays the structure of the table created by **enumUIObjectProperties**:

Field name	Type	Description
ObjectName	A128	Name of the object
PropertyName	A64	Name of the property
PropertyType	A48	Data type of the corresponding property
PropertyValue	A255	Value of the corresponding property

In Syntax 2, the properties of the object (but *not* the properties of objects it contains) are written to a DynArray named *properties*. The DynArray keys are the property names, and the items are the corresponding values. You must declare the DynArray before calling **enumUIObjectProperties** .

Examples

{button ,AL(`OPAL_TYPE_UIOBJECT;OPAL_METH_UIEONT;OPAL_METH_UIECTT;',0,"Defaultoverview",)}

Related Topics

enumUIObjectProperties method examples

Example1 Writing property values to a table

Example2 Writing property values to the dynamic array

enumUIObjectProperties example 1

The following example assumes that *getProperties* is a button on a form designed to display fields from the *Customer* table. The **pushButton** method for *getProperties* uses **enumUIObjectProperties** to write all of the property values for each object on the form to a table named *CstProps*.

```
; getProperties::pushButton
method pushButton(var eventInfo Event)
   enumUIObjectProperties("CstProps.db")
endMethod
```

enumUIObjectProperties example 2

The following example assumes that *getProperties* is a button on a form. The **pushButton** method for *btnProperties* uses **enumUIObjectProperties** to write all of its property values to a dynamic array named *dyn* and then display it.

```
; btnProperties::pushButton
method pushButton(var eventInfo Event)
   var
      dyn   DynArray[] String
   endVar


   self.enumUIObjectProperties(dyn)
   dyn.view("Properties of this button:")
endMethod
```

execMethod method/procedure

Calls a custom method that takes no arguments.

Syntax

```
execMethod ( const methodName String )
```

Description

execMethod calls the custom method specified by *methodName*. The method specified in *methodName* takes no arguments. Because **execMethod** allows you to call a method based on the contents of a variable, the compiler does not know which method to call until run time.

Example

{button ,AL(`OPAL_TYPE_UIOBJECT;OPAL_METH_UIMETHODDELETE;OPAL_METH_UIMETHODGET;OPAL_METH_UIMETHODSET;',0,"Defaultoverview",)} Related Topics

execMethod example

The following examples assumes that a form contains three fields: *fieldOne*, *fieldTwo*, and *fieldThree*. The form's Var window declares a dynamic array named *objPreProc*, and the form's custom method is named *fieldOnePreProc*. The form's **open** method (which appears in the **isPreFilter=False** clause) creates elements in the *objPreProc* array. An element is created for each object on the form that has a preprocessing custom method.

In the following example, *fieldOne* is assumed to require some preprocessing. An array element is created with the index *pageOne*.*fieldOne*, and the custom method name *fieldOnePreProc*. The **isPreFilter=True** clause is called for each object on the form to determine whether an array element in *objPreProc* corresponds to the active object. If so, the custom method for that object is called.

The following code attaches to the custom method **fieldOnePreProc**:

```
; form design::fieldOnePreProc  (custom method)
; This method is called during the form's preFilter clause,
; when the current object is fieldOne.
method fieldOnePreProc()
fieldOne.color = "Red"    ; change the color of the field
fieldOne.Value = "Initialized by the form's open method"
endMethod
```

The following code goes in the form's Var window:

```
; Var window for the form
Var
  ObjPreProc DynArray[] String  ; indexed by object name, will
                                ; hold names of methods to execute
                                ; when isPreFilter is true
endVar
```

The following code attaches to the form's **open** method:

```
method open(var eventInfo Event)
var
  targObj   UIObject   ; holds the target object
  targName  String     ; target object's name
  element   AnyType    ; index to dynamic array objPreProcs
endVar
if eventInfo.isPreFilter()
  then
    ; code here executes for each object in form
    eventInfo.getTarget(targObj)  ; identify the current target
    targName = targObj.name       ; retrieve the name of the target
    forEach element in objPreProc ; iterate through array
      if element = targName then  ; is the target name there?
                                  ; if so, execute the corresponding
                                  ; custom method
        execMethod(objPreProc[targName])
      endIf
    endForEach
  else
    ; code here executes just for form itself


    ; assign elements to the objPreProc array to indicate
    ; objects for which there is a preprocess custom method
    objPreProc["fieldOne"] = "fieldOnePreProc"
endIf
endMethod
```

forceRefresh method

Instructs an object to display the specified data in the underlying table, and causes a calculated field to recalculate.

Syntax

forceRefresh ( ) Logical

Description

forceRefresh instructs an object to display the specified data in the underlying table, and causes a calculated field to recalculate. **forceRefresh** also causes a calculated field to recalculate its value, and causes a crosstab or chart to re-evaluate its components.

Calling **active.forceRefresh( )** is the same as calling **active.action(DataRecalc)** or pressing SHIFT + F9. **active.forceRefresh( )** is a UIObject counterpart to the **forceRefresh** method defined for the TCursor type.

A call to **forceRefresh** affects the target object, objects contained by the target object, and objects bound to the same table as the target object. This method does not affect objects in other windows. For example, calling **forceRefresh** in a form does not refresh data displayed in a table window. Refresh each object in a form by declaring a UIObject variable and calling attach to assign it a value. Do not use a variable declared as a Form variable.

forceRefresh behaves as follows:

- If a table frame or MRO is active when you call **forceRefresh**, only the underlying table refreshes. Child tables repaint, but do not discard cached data.
- If a field object is active when you call **forceRefresh**, the table associated with that field refreshes, and all fields dependent on it are repainted.
- You will not lose your active record position, provided the record still exists in the table.
- On an SQL server, a call to **forceRefresh** forces a read from the server. This is the only way to get a refresh from the server. **forceRefresh** only works on an SQL table if the table has a unique index.

Example

{button ,AL(`OPAL_TYPE_UIOBJECT;OPAL_METH_UICURR;OPAL_METH_TCFORCEREFRESH;',0,"Defaultoverview",)} Related Topics

forceRefresh example

The following example uses **forceRefresh** in code attached to a button's built-in **pushButton** method, allowing the user to control when data is refreshed. This example assumes you have interactively chosen the Database page from the Preferences tabbed dialog box and entered a large value (at least 3,600 seconds) in the Refresh rate dialog box. This code uses **forceRefresh** to refresh the Parts table each time the user clicks the button. Other tables that are bound to this form are refreshed once every 3,600 seconds (one hour).

```
method pushButton(var eventInfo Event)
   Parts.forceRefresh()
endMethod
```

getBoundingBox method

Returns the coordinates of the frame that bounds an object.

Syntax

```
getBoundingBox ( var topLeft Point, var bottomRight Point )
```

Description

getBoundingBox returns the coordinates of the top left corner (*topLeft*) and the bottom right corner (*bottomRight*) of the frame that bounds an object. The coordinates are specified relative to the form. When you select an object in the design window, its bounding box is visible.

Example

{button ,AL(`OPAL_TYPE_UIOBJECT;OPAL_METH_UICONV;',0,"Defaultoverview",)} Related Topics

getBoundingBox example

The following example draws a box around an ellipse based on the ellipse's bounding box. Assume that a form contains an ellipse named *redCircle*.

```
; redCircle::mouseUp
method mouseUp(var eventInfo MouseEvent)
var
  TopLeft,
  BotRight Point      ; to hold the points returned by getBoundingBox
  ui       UIObject   ; to create a new object
endVar


self.getBoundingBox(TopLeft, BotRight)
ui.create(BoxTool, TopLeft.x(),
                   TopLeft.y(),
                   BotRight.x() - TopLeft.x(),
                   BotRight.y() - TopLeft.y())
ui.Color = Green
ui.Translucent = Yes
ui.Visible = Yes


endMethod
```

getGenFilter method

Retrieves the filter criteria associated with a field, table frame, or multi-record object.

Syntax

```
1. getGenFilter ( criteria DynArray[ ] AnyType ) Logical

2. getGenFilter ( criteria Array[ ] AnyType [ , fieldName Array[ ] AnyType ] ) Logical

3. getGenFilter ( criteria String ) Logical
```

Description

getGenFilter retrieves the filter criteria associated with a field, table frame, or multi-record object. **getGenFilter** assigns them to a DynArray variable (Syntax 1) or to two Array variables (Syntax 2) that you declare and include as arguments. Values are not returned directly.

In Syntax 1, a dynamic array (DynArray) named *criteria* lists fields and filtering conditions as follows: the index is the field name, and the item is the corresponding filter expression.

In Syntax 2, an Array named *criteria* lists filtering conditions, and the optional Array *fieldName* lists corresponding field names. If you omit *fieldName*, conditions apply to fields in the order they appear in the *criteria* array. The first condition applies to the first field in the table, the second condition applies to the second field, and so on.

If the arrays used in Syntax 2 are resizeable, **getGenFilter** adjusts the array size to equal the number of fields in the underlying table. If fixed-size arrays are used, this method stores as many criteria as it can, starting with criteria field 1. If there are more array items than fields, the remaining items are left empty. If there are more fields than items, this method fills the array and then stops.

In Syntax 3, the filter criteria is assigned to a String variable named *criteria.* You must declare and pass *criteria* as an argument.

Examples

{button ,AL(`OPAL_TYPE_UIOBJECT;OPAL_METH_UISETGENFILTER;OPAL_METH_UIDROPGENFILTER;OPAL_METH_UISETRANGE;OPAL_METH_TCGETGENFILTER;OPAL_INFO_TBUSINGRANGESANDFILTERS;',0,"Default overview",)} **Related Topics**

getGenFilter method examples

Example1 Populating a dynamic array (DynArray) with a table frame's criteria

Example2 Tracking the current filter criteria without setting flags

getGenFilter example 1

In the following example, the **pushButton** method for a button named *btnSetFilter* uses **getGenFilter** to populate a dynamic array (DynArray) named *dyn* with a table frame's filter criteria. The code then it examines the DynArray to see if the current criteria filters the Balance Due field for values greater than 10,000 and the Total Invoice field for values less than 65,000 and resets the filter if necessary.

```
;btnSetFilter :: pushButton
method pushButton(var eventInfo Event)
   var
      currentDyn,
      filterDyn   DynArray[] AnyType
      keysAr      Array[] AnyType
   endVar


   filterDyn["Balance Due"]   = "> 10000"
   filterDyn["Total Invoice"] = "< 65000"


   ORDERS.getGenFilter(currentDyn)   ; ORDERS is a table frame on a form.


   if currentDyn = filterDyn then
         return                         ; Filter is OK.
   else
         ORDERS.setGenFilter(filterDyn) ; Reset filter.
   endIf
endMethod
```

getGenFilter example 2

In the following example, the **pushButton** method for a button named *btnShowFilter* uses **getGenFilter** to populate a dynamic array (DynArray) named *dyn* with the current filter criteria. The code then displays the DynArray in a **view** dialog box. Use this technique as an alternative to setting flags to track the current filter criteria.

```
;btnShowFilter :: pushButton
method pushButton(var eventInfo Event)
   var
      dyn   DynArray[] AnyType
   endVar


   ORDERS.getGenFilter(dyn)   ; ORDERS is a table frame on a form.
   dyn.view("Current filter criteria")
endMethod
```

getHTMLTemplate method

Returns the HTML string of the UIObject.

Syntax

```
getHTMLTemplate ( ) String
```

Description

getHTMLTemplate generates source HTML for a UIObject and returns it as a String.

Example

{button ,AL(`OPAL_TYPE_UIOBJECT;;',0,"Defaultoverview",)} **Related Topics**

getHTMLTemplate example

In the following example, code in a script window manipulates a customer form by associating a form variable with the form, attaching to a particular field object on the form and generating a string containing the HTML source for this field. The HTML source is displayed in a window. Assume that the customer form contains the customer table in its data model. Fields on the form (including Customer No) are bound to corresponding fields in the table.

```
method run(var eventInfo Event)
var
   fm      Form
   UIO     UIObject
   stHTML  String
endVar

fm.open("customer.fsl")
UIO.attach(fm.Customer_No)      ; attach the UIO variable to the
                                ; Customer_No field of the customer
                                ; table
stHTML=UIO.getHTMLTemplate()
stHTML.view()                   ; displays: <TABLE><TR><TD>Customer No:
                                ; </TD><TD><INPUT TYPE="TEXT"
                                ; NAME="Customer_No"></TD></TR></TABLE>
endMethod
```

getPosition method

Reports the position of an object on the screen.

Syntax

```
getPosition ( const x LongInt, const y LongInt, const w LongInt, const h LongInt)
```

Description

getPosition retrieves the position of an object on the screen, relative to its container. Variables *x* and *y* specify the coordinates (in twips) of the upper-left corner of the object. Variables *w* and *h* specify the object's width and height (in twips). If the object is not specified, *self* is implied.

To ObjectPAL, the screen is a two-dimensional grid, with the origin (0, 0) at the upper-left corner of an object's container, positive x-values extending to the right, and positive y-values extending down.

For dialog boxes and for the Corel Paradox desktop application, the object's position is specifed relative to the entire screen; for forms, reports, and table windows, the position is specified relative to the Corel Paradox desktop.

Example

{button ,AL(`OPAL_TYPE_UIOBJECT;OPAL_METH_UISPOS;',0,"Defaultoverview",)} Related Topics

getPosition example

The following example moves a circle across the screen in response to timer events. The **pushButton** method for *toggleButton* uses **setTimer** and **killTimer** to start or stop a timer, depending on the button's position. When the timer starts, it issues a timer event every 100 milliseconds. Each **timer** event causes *toggleButton's* timer method to execute. The timer method locates the current position of the circle using **getPosition**, and moves it 100 twips to the right using **setPosition**.

The following code attaches to *toggleButton*'s **pushButton** method:

```
; toggleButton::pushButton
method pushButton(var eventInfo Event)
if buttonLabel = "Start Timer" then  ; if stopped, then start
  buttonLabel = "Stop Timer"         ; change label
  self.setTimer(100)          ; tell timer to issue a timer
                              ; event every 100 milliseconds
else
  buttonLabel = "Start Timer"         ; change label
  self.killTimer()                    ; stop the timer
endIf


endMethod
```

The following code attaches to *toggleButton*'s **timer** method:

```
; toggleButton::timer
;   this method is called once for every timer event
method timer(var eventInfo TimerEvent)
var
  ui          UIObject
  x, y, w, h  SmallInt
endVar


ui.attach(floatCircle)          ; attach to the circle
ui.getPosition(x, y, w, h)      ; assign coordinates to vars
if x < 4320 then                ; if not at right edge of area
  ui.setPosition(x + 100, y, w, h)  ; move to the right
else
  ui.setPosition(1440, y, w, h)     ; return to the left
endIf


endMethod
```

getProperty method

Returns the value of a specified property.

Syntax

```
getProperty ( const propertyName String ) AnyType
```

Description

getProperty returns the value of the property specified in *propertyName*. Not all properties take strings as values. For example, if a property value is a number, this method returns a number. To return a string in each case, use **getPropertyAsString**.

Use **getProperty** when *propertyName* is a variable as an alternative to retrieving a property directly. Otherwise, access the property directly/ The following code displays the syntax for getting property directly:

```
thisColor = myBox.Color
```

Example

**{button ,AL(`OPAL_TYPE_UIOBJECT;OPAL_METH_UIGPROST;OPAL_METH_UISPRO;',0,"Defaultoverview",)
} Related Topics**

getProperty example

The following example creates a dynamic array that is indexed by property names and contains property values. The array's index is used as the argument to the **getProperty** command.

```
; boxOne::mouseUp
method mouseUp(var eventInfo MouseEvent)
var
  propNames DynArray[] AnyType    ; to hold property names & values
  arrayIndex           String     ; index to dynamic array
endVar


propNames["Color"] = ""
propNames["Visible"] = ""
propNames["Name"] = ""


foreach arrayIndex in propNames   ; assign the properties to the array
  propNames[arrayIndex] = self.getProperty(arrayIndex)
endforeach


propNames["Color"] = "DarkBlue"


foreach arrayIndex in propNames   ; set properties from the array
  self.setProperty(arrayIndex, propNames[arrayIndex])
endforeach


endMethod
```

getPropertyAsString method

Returns the value of a specified property as a string.

Syntax

```
getPropertyAsString ( const propertyName String ) String
```

Description

getPropertyAsString returns a string containing the value of the property specified in *propertyName*.

Example

{button ,AL(`OPAL_TYPE_UIOBJECT;OPAL_METH_UIGPRO;OPAL_METH_UISPRO;',0,"Defaultoverview",)}

Related Topics

getPropertyAsString example

The following example assigns the value of the Color property to an AnyType variable. The value returned is a LongInt, because colors are long integer constants. Next, the Color property is obtained using **getPropertyAsString**. The value returned is a String type (e.g. Blue).

```
; boxOne::mouseRightUp
method mouseRightUp(var eventInfo MouseEvent)
var
  myColor  AnyType
endVar


myColor = self.getProperty("Color")
myColor.view()                              ; shows as LongInt
myColor = self.getPropertyAsString("Color")
myColor.view()                              ; shows as String
endMethod
```

getRange method

Retrieves the values that specify a range for a field, table frame, or multi-record object.

Syntax

```
getRange ( var rangeVals Array[ ] String ) Logical
```

Description

getRange retrieves the values that specify a range for a field, table frame, or multi-record object. This method assigns values to an Array variable that you declare and include as an argument. The following table displays the array values and their range criteria:

Number of array items	Range specification
No items (empty array)	No range criteria is associated with the UIObject
One item	Specifies a value for an exact match on the index's first field
Two items	Specifies a range for the index's first field
Three items	The first item specifies an exact match for the index's first field; items 2 and 3 specify a range for the index's second field
More than three items	For an array of size *n*, specify exact matches on the index's *n*-2 fields. The last two array items specify a range for the index's *n*-1 fields.

If the array is resizeable, **getRange** sets the size to equal the number of fields in the underlying table. If fixed-size arrays are used, this method stores as many criteria as it can, starting with criteria field 1. If there are more array items than fields, the remaining items are left empty; if there are more fields than items, this method fills the array and then stops.

Example

{button ,AL(`OPAL_TYPE_UIOBJECT;OPAL_METH_UISETRANGE;OPAL_METH_UIGETGENFILTER;OPAL_METH_TBGETRANGE;OPAL_METH_TCGETRANGE;OPAL_INFO_TBUSINGRANGESANDFILTERS;',0,"Defaultoverview",)} **Related Topics**

getRange example

The following example uses ObjectPAL to link two unlinked tables in the data model. Assume that a form has the *Orders* and *Lineitem* tables in its data model and they are not linked. **getRange** is used on a table frame bound to the *Lineitem* table to retrieve the values that specify the current range.

The following code is attached to the record object's **arrive** method of a table frame that is bound to the *Orders* table:

```
;Record :: arrive
method arrive(var eventInfo MoveEvent)
   var
      arSet   Array[] AnyType
      arGet   Array[] AnyType
   endVar


   LINEITEM.getRange(arGet)      ;Retrieve values of range.


   arSet.setSize(2)         ;Specify size of array.
   arSet[1] = string(Order_No.value)
   arSet[2] = string(Order_No.value)


   if (arSet.size() = arGet.size()) and (arSet <> arGet) then
      LINEITEM.setRange(arSet)      ;Specify range of records.
   endIf
endMethod
```

getRGB procedure

Returns the red, green, and blue components of a color.

Syntax

```
getRGB ( const rgb LongInt, var red SmallInt, var green SmallInt, var blue SmallInt )
```

Description

getRGB returns the component red, green and blue components of the color specified in *rgb*. *rgb* is a Colors constant. **getRGB** assigns the component values to the variables *red*, *green*, and *blue*. You must declare and pass the *red*, *green*, and *blue* variables as arguments.

Example

{button ,AL(`OPAL_TYPE_UIOBJECT;OPAL_METH_UIRGB;',0,"Defaultoverview",)} Related Topics

getRGB example

The following example determines the red, green, and blue components of the constant Brown.

```
; decompBrown::pushButton
method pushButton(var eventInfo Event)
var
  thisRed, thisBlue, thisGreen SmallInt
endVar
getRGB(Brown, thisRed, thisGreen, thisBlue)
msgInfo("Brown is really",
        String("Red ", thisRed, "  Green ", thisGreen,
               "  Blue ", thisBlue))
endMethod
```

hasMouse method

Determines whether the pointer is positioned over an object.

Syntax

hasMouse ( ) Logical

Description

hasMouse returns True if the pointer is positioned within the boundaries of an object; otherwise, it returns False.

Example

{button ,AL(`OPAL_TYPE_UIOBJECT;OPAL_TYPE_MOUSEEVENT;',0,"Defaultoverview",)} **Related Topics**

hasMouse example

The following example assumes that a form has a bitmap object named *cat*. The **open** method for *cat* sets a timer interval to 250 milliseconds. The **timer** method uses **hasMouse** to determine if the mouse is within *cat*'s boundaries; if not, it moves *cat* to the mouse's position.

The following code attaches to *cat*'s **open** method:

```
; cat::open
method open(var eventInfo Event)
; set the timer interval to 250 milliseconds
self.setTimer(250)
endMethod
```

The following code attaches to *cat*'s **timer** method:

```
; cat::timer
method timer(var eventInfo TimerEvent)
var
  mousePt  Point                    ; to get mouse position
endVar
if NOT cat.hasMouse() then          ; am I on the mouse?
  mousePt = getMouseScreenPosition() ; find the mouse
  cat.setPosition(mousePt.x() - 350,
                  mousePt.y() - 2880,
                  4320, 1750)       ; chase the mouse
  ; moves cat above and slightly to the left of mouse
  ; assumes cat is a bitmap with width 4320, height 1750
  ; since getMouseScreenPosition returns position of mouse
  ; on desktop, these numbers assume form is maximized
  ; offset (2880-1750) allows for height of menu and Toolbar
endIf
endMethod
```

home method

Moves to the first record in a table.

Syntax

home ( ) Logical

Description

home sets the active record to the first record in a table. **home** respects the limits of restricted views that are displayed in a linked table frame or multi-record object. **home** moves to the first record in a restricted view.

home has the same effect as the action constant DataBegin. This means that the following statements are equivalent:

```
obj.home()
obj.action(DataBegin)
```

Example

{button ,AL(`OPAL_TYPE_UIOBJECT;OPAL_METH_UIEND;OPAL_METH_UINEXTR;OPAL_METH_UIPRIO;',0,"Defaultoverview",)} Related Topics

home example

The following example moves to the first record in the *Customer* table. Assume that *Customer* is bound to a table frame on the form, and *moveToHome* is a button on the form.

```
; moveToHome::pushButton
method pushButton(var eventInfo Event)
CUSTOMER.home()  ; move to the first record
                 ; same as:  CUSTOMER.action(DataBegin)
msgInfo("At the first record?", CUSTOMER.atFirst())
endMethod
```

insertAfterRecord method

Inserts a record below the active record in a table.

Syntax

```
insertAfterRecord ( ) Logical
```

Description

insertAfterRecord inserts a record below the active record in a table. The table must be in Edit mode.

Example

{button ,AL(`OPAL_TYPE_UIOBJECT;OPAL_METH_UIIREC;OPAL_METH_UIIBEF;',0,"Defaultoverview",)}

Related Topics

insertAfterRecord example

The following example assumes that *CustSort* is a copy of the *Customer* table that has been sorted by the Name field. The form in this example contains a table frame named *CUSTSORT* that is bound to the *CustSort* table, an undefined field named *newField*, and a button named *insRecButton*. To add a name to the table, type the name in *newField* and press *insRecButton*.

The following code is attached to the **pushButton** method for *insRecButton*. This method determines if a value has been added to *newField* and if the form is in Edit mode. The method attaches the TCursor *custTC* to *CUSTSORT*, and scans *custTC* for a value greater than the string given in *newField*. If it detects a name greater than the new name, the method uses **insertRecord** to add a blank record before the name found; otherwise, it uses **insertAfterRecord** to insert a new blank record to the end of the table.

```
; insRecButton::pushButton
method pushButton(var eventInfo Event)
var
  custTC  TCursor
  nameStr String
endvar


if newField.Value = "" then       ; Quit if the field is blank.
  RETURN
endIf


nameStr = newField.Value          ; Get the name to add.
CUSTSORT."Name".moveTo()


if CUSTSORT.isEdit() then          ; Check for edit mode first.
  custTC.attach(CUSTSORT)


  scan custTC for custTC."Name" >= nameStr:
    quitloop                      ; Stop when you find the name.
  endscan


  msgInfo("Active record no", custTC.recno())
  CUSTSORT.resync(custTC)         ; Resync CUSTSORT to custTC.


  if NOT CUSTSORT.atLast() then
    CUSTSORT.insertBeforeRecord()
  else
    CUSTSORT.insertAfterRecord()  ; Add blank record.
  endIf



  CUSTSORT.Name=newField.Value

  CustSort.postRecord()

  msgInfo("New name added", "Please enter remaining customer information")


else
  msgInfo("Sorry", "Form must be in Edit mode.")
endIf
endMethod
```

insertBeforeRecord method

Inserts a record above the active record in a table.

Syntax

```
insertBeforeRecord ( ) Logical
```

Description

insertBeforeRecord inserts a record above the active record in a table. The table must be in Edit mode.

insertBeforeRecord has the same effect as the action constant DataInsertRecord. This means the following statements are equivalent:

```
obj.insertBeforeRecord()
obj.action(DataInsertRecord)
```

Example

{button ,AL(`OPAL_TYPE_UIOBJECT;OPAL_METH_UIIREC;OPAL_METH_UIIAFT;',0,"Defaultoverview",)}

Related Topics

insertBeforeRecord example

The following example assumes that *CustSort* is a copy of the *Customer* table that has been sorted by the Name field. The form contains a table frame named *CUSTSORT* that is bound to *CustSort*, an undefined field named *newField*, and a button named *insRecButton*. To add a name to the table, type the name in *newField* and press *insRecButton*.

The following method overrides the **pushButton** method for *insRecButton*. This method determines if a value has been added to *newField* and if the form is in Edit mode. The method attaches a TCursor named *custTC* to *CUSTSORT*, and scans *custTC* for a value greater than the string given in *newField*. If the method detects a name greater than the new name, the method uses **insertBeforeRecord** to insert a blank record before the name found; otherwise, it uses **insertAfterRecord** to insert a new blank record at the end of the table.

```
; insRecButton::pushButton
method pushButton(var eventInfo Event)
var
  custTC  TCursor
  nameStr String
endvar


if newField.Value = "" then       ; Quit if the field is blank.
  RETURN
endIf


nameStr = newField.Value          ; Get the name to add.
CUSTSORT."Name".moveTo()


if thisForm.Editing then          ; Check for edit mode first.
  custTC.attach(CUSTSORT)


  scan custTC for custTC."Name" >= nameStr:
    quitloop                      ; Stop when you find the name.
  endscan


  msgInfo("Active record no", custTC.recno())
  CUSTSORT.resync(custTC)         ; Resync CUSTSORT to custTC.


  if NOT CUSTSORT.atLast() then
    CUSTSORT.insertBeforeRecord()
  else
    CUSTSORT.insertAfterRecord()
  endIf


  ; ... fill the record with the rest of the customer information
; Put new name in the field of the tableframe and post. Inform user.

  CUSTSORT.Name=newField.Value

  CustSort.postRecord()

  msgInfo("New name added", "Please enter remaining customer information")


else
  msgInfo("Sorry", "Form must be in Edit mode.")
endIf
endMethod
```

insertRecord method

Inserts a record before the active record in a table.

Syntax

```
insertRecord ( ) Logical
```

Description

insertRecord inserts a record before the active record in a table.

insertRecord has the same effect as **insertBeforeRecord** and the action constant DataInsertRecord. This means the following three statements are equivalent:

```
obj.insertRecord()
obj.insertBeforeRecord()
obj.action(DataInsertRecord)
```

Example

{button ,AL(`OPAL_TYPE_UIOBJECT;OPAL_METH_UIIAFT;OPAL_METH_UIIBEF;',0,"Defaultoverview",)}

Related Topics

insertRecord example

See the **insertBeforeRecord** example.

isContainerValid method

Reports whether an object's container is valid.

Syntax

isContainerValid ( ) Logical

Description

isContainerValid reports if the active object's container is valid. For example, if a form does not have a container the ContainerName property for a form is not valid.

Example

{button ,AL(`OPAL_TYPE_UIOBJECT;OPAL_METH_UIISLMC;',0,"Defaultoverview",)} **Related Topics**

isContainerValid example

In the following example, the **arrive** built-in event method for a form uses **isContainerValid** to search for a valid container:

```
; thisForm::arrive
method arrive(var eventInfo MoveEvent)
  if eventInfo.isPreFilter() then


     ;Code here executes before each object
  else
     ;Code here executes afterwards (or for form)
     if NOT isContainerValid() then
       msgInfo("Form",
               "This object does not have a valid container.")
     endIf
  endIf
endMethod
```

isEdit method

Reports whether an object is in Edit mode.

Syntax

isEdit ( ) Logical

Description

isEdit reports whether an object is in Edit mode.

Example

{button ,AL(`OPAL_TYPE_UIOBJECT;OPAL_METH_UIEDIT;OPAL_METH_UIECED;OPAL_METH_UILOCKRECORD;',0,"Defaultoverview",)} **Related Topics**

isEdit example

See the **lockRecord** example.

isEmpty method

Reports whether a table contains records.

Syntax

```
isEmpty ( ) Logical
```

Description

isEmpty returns True if none of the table's records are associated with the table frame. **isEmpty** respects the limits of restricted views displayed in a linked table frame or multi-record object.

You can also determine if a table is empty by determining the value returned by the **nRecords** method or the value of the object's NRecords property.

Example

{button ,AL(`OPAL_TYPE_UIOBJECT;OPAL_METH_UIEMP;OPAL_METH_UINREC;',0,"Defaultoverview",)}

Related Topics

isEmpty example

The following example uses the *cascadeDelete* button to delete an order and all the linked detail records for that order. Assume that a form contains a single-record object that is bound to the *Orders* tables and a linked table frame that is bound to the *Lineitem* table. *Orders* has a one-to-many link to *Lineitem*.

```
; cascadeDelete::pushButton
method pushButton(var eventInfo Event)
var
  ui        UIObject
endVar


if thisForm.Editing then
  if msgQuestion("Confirm", "Delete this order?") = "Yes" then
    ui.attach(LINEITEM)
    while NOT ui.isEmpty()      ; check to see if linked table is
                                ; empty respects restricted view
      ui.deleteRecord()         ; delete the detail records
    endwhile
    ORDERS.action(DataDeleteRecord)   ; delete the master record
  endIf
else
  msgInfo("Status", "You must be editing to delete a record.")
endIf
endMethod
```

isLastMouseClickedValid method

Reports whether the last object clicked is valid.

Syntax

isLastMouseClickedValid ( ) Logical

Description

isLastMouseClickedValid reports whether the active form has been clicked since it opened.

Example

{button ,AL(`OPAL_TYPE_UIOBJECT;OPAL_METH_UIISLMR;',0,"Defaultoverview",)} **Related Topics**

isLastMouseClickedValid example

The following example determines whether a form has been clicked:

```
; thisForm::arrive
method arrive(var eventInfo MoveEvent)
   if eventInfo.isPreFilter() then
     ;Code here executes before each object
  else
     ;Code here executes afterwards (or for form)
     if NOT isLastMouseClickedValid() then
       msgInfo("FYI", "This form has not been clicked yet.")
     endIf
  endIf
endMethod
```

isLastMouseRightClickedValid method

Reports whether the last object right-clicked is valid.

Syntax

isLastMouseRightClickedValid ( ) Logical

Description

isLastMouseRightClickedValid reports whether the current form has been right-clicked since it opened.

Example

{button ,AL(`OPAL_TYPE_UIOBJECT;OPAL_METH_UIISLMC;',0,"Defaultoverview",)} **Related Topics**

isLastMouseRightClickedValid example

The following example determines whether a form has been right-clicked:

```
; thisForm::arrive
method arrive(var eventInfo MoveEvent)
  if eventInfo.isPreFilter() then

     ;Code here executes before each object
  else
     ;Code here executes afterwards (or for form)
     if NOT isLastMouseRightClickedValid() then
       msgInfo("FYI", "This form has not been right-clicked yet.")
     endIf
  endIf
endMethod
```

isRecordDeleted method

Reports whether the active record has been deleted (dBASE tables only).

Syntax

```
isRecordDeleted ( ) Logical
```

Description

isRecordDeleted reports whether the active record has been deleted. **isRecordDeleted** only works for dBASE tables. Deleted Corel Paradox records can't be displayed. This method returns True if the active record has been deleted; otherwise, it returns False.

For **isRecordDeleted** to work correctly, you must call **showDeleted** (TCursor type) to display deleted records in the table; otherwise, deleted records are not visible to **isRecordDeleted**.

Example

{button ,AL(`OPAL_TYPE_UIOBJECT;OPAL_METH_UIUNDELETERECORD;OPAL_METH_TCISSHOWDELETEDON;OPAL_METH_TCSHDEL;',0,"Defaultoverview",)} Related Topics

isRecordDeleted example

See the **isRecordDeleted** (Tcursor type) example.

keyChar method

Sends an event to an object's keyChar method.

Syntax

```
1. keyChar ( const characters String [ , const state SmallInt ] ) Logical
2. keyChar ( const ansiKeyValue SmallInt ) Logical
3. keyChar ( const ansiKeyValue SmallInt, const vChar SmallInt, const state SmallInt ) Logical
```

Description

keyChar creates an event and to call the object's built-in **keyChar** event method. Specify one or more characters in *characters* (Syntax 1), in *ansiKeyValue* (Syntax 2), or in *ansiKeyValue* and *vChar* (Syntax 3). Specify the keyboard state in *state* using KeyboardStates constants. You can add these constants together to create combined key states (e.g., ALT + CTRL).

Example

{button ,AL(`OPAL_TYPE_UIOBJECT;OPAL_METH_UIKEPH;OPAL_TYPE_KEYEVENT;',0,"Defaultoverview",)}

Related Topics

keyChar example

The following example overrides the **pushButton** method of a button named *sendKeyChar*. This method sends keystrokes *fieldOne* on the form.

```
; sendKeyChar::pushButton
method pushButton(var eventInfo Event)
var
  x  SmallInt
endVar
fieldOne.keyChar("Send me an ")    ; send a string
fieldOne.keyChar(65, 65, Shift)    ; send ANSI char, decimal
                                   ; equivalent of VK_Char,
                                   ; and keyboardstate
fieldOne.keyChar(" and a ", Shift) ; send a string with the keyboardstate
x = 98                             ; set the code
fieldOne.keyChar(x)                ; send ANSI char code
endMethod
```

keyPhysical method

Sends an event to an object's built-in **keyPhysical** method.

Syntax

```
keyPhysical ( const aChar SmallInt, const vChar SmallInt, const state SmallInt )
```

Description

keyPhysical sends an event to an object's built-in **keyPhysical** method. Specify the ANSI character code in *aChar*, the virtual key code in *vChar*, and the keyboard state in *state* using KeyboardStates constants.

Example

{button ,AL(`OPAL_TYPE_UIOBJECT;OPAL_METH_UIKECH;OPAL_TYPE_KEYEVENT;',0,"Defaultoverview",)}

Related Topics

keyPhysical example

In the following example, code is attached to the **pushButton** method of a button named *sendKeyPhys*. This method sends the character a to *fieldOne*.

```
; sendKeyPhys::pushButton
method pushButton(var eventInfo Event)
   fieldOne.keyPhysical(97, 97, Shift)   ; send an "a"
endMethod
```

killTimer method

Stops the timer associated with an object.

Syntax

```
killTimer ( )
```

Description

killTimer stops the timer associated with an object.

Example

{button ,AL(`OPAL_TYPE_UIOBJECT;OPAL_METH_UISTIM;',0,"Defaultoverview",)} **Related Topics**

killTimer example

The following example moves a circle across the screen in response to TimerEvents. The **pushButton** method for *toggleButton* uses **setTimer** and **killTimer** to start or stop a timer respectively. When the timer starts, it issues a TimerEvent every 100 milliseconds. Each TimerEvent causes *toggleButton's* **timer** method to execute. The **timer** method uses **getPosition** to retrieve the current position of the ellipse and uses **setPosition** to move it 100 twips to the right.

The following code is attached to *toggleButton*'s **pushButton** method:

```
; toggleButton::pushButton
method pushButton(var eventInfo Event)
if buttonLabel = "Start Timer" then  ; if stopped and start
  buttonLabel = "Stop Timer"         ; change label
  self.setTimer(100)          ; tell timer to issue a timer
                              ; event every 100 milliseconds
else
  buttonLabel = "Start Timer"         ; change label
  self.killTimer()                    ; stop the timer
endIf


endMethod
```

The following code is attached to *toggleButton*'s **timer** method. FloatCircle is a circle UI Object on the form:

```
; toggleButton::timer
;   this method is called once for every timer event
method timer(var eventInfo TimerEvent)
var
  ui          UIObject
  x, y, w, h  SmallInt
endVar


ui.attach(floatCircle)          ; attach to the circle
ui.getPosition(x, y, w, h)      ; assign coordinates to vars
if x < 4320 then                ; if not at right edge of area
  ui.setPosition(x + 100, y, w, h)  ; move to the right
else
  ui.setPosition(1440, y, w, h)     ; return to the left
endIf


endMethod
```

locate method

Searches for a specified field value.

Syntax

```
1. locate ( const fieldName String, const exactMatch AnyType [ ,const fieldName String, const
exactMatch AnyType ] * ) Logical

2. locate ( const fieldNum SmallInt, const exactMatch AnyType [ ,const fieldNum SmallInt, const
exactMatch AnyType ] * ) Logical
```

Description

locate searches a table frame, multi-record object, record object, or field object for record values that match one or more field/value pairs. Specify the search value in *exactMatch* and the search field in *fieldName* or *fieldNum* (use *fieldNum* for faster performance). When possible, **locate** uses active indexes to speed the search. This method respects the limits of restricted views in linked detail tables.

If a match is found, the cursor moves to that record. This operation fails if the active record cannot be posted and unlocked (e.g., due to a key violation). If no match is found, the cursor returns to the active record. The search always starts from the beginning of the table.

Note

- The search is case-sensitive unless **ignoreCaseInLocate** (Session type) is enabled.

Example

{button ,AL(`OPAL_TYPE_UIOBJECT;OPAL_METH_UILOCATENEXT;OPAL_METH_UILOCATEPATTERN;OPAL_METH_UILOCATENEXTPATTERN;',0,"Defaultoverview",)} Related Topics

locate example

The following example assumes that a form contains a table frame bound to the *Customer* table and a button named *locateButton*. The **pushButton** method for *locateButton* searches for the customer named Sight Diver in the city named Kato Paphos. If a match is found, the customer's name is changed to Right Diver.

```
; locateButton::pushButton
method pushButton(var eventInfo Event)
var
  Cust UIObject
endVar
Cust.attach(CUSTOMER)
; find customer named "Sight Diver" in Kato Paphos
if Cust.locate("Name", "Sight Diver", "City", "Kato Paphos") then
   Cust.edit()
   Cust."Name" = "Right Diver"
   Cust.endEdit()
endIf
endMethod
```

locateNext method

Searches forward from the active record for a specified field value.

Syntax

```
1. locateNext ( const fieldName String, const exactMatch AnyType [ , const fieldName String,
const exactMatch AnyType ] * ) Logical

2. locateNext ( const fieldNum SmallInt, const exactMatch AnyType [ , const fieldNum SmallInt,
const exactMatch AnyType ] * ) Logical
```

Description

locateNext searches a table for record values that match one or more field/value pairs. Specify the search value in *exactMatch* and the search field in *fieldName* or *fieldNum* (use *fieldNum* for faster performance). When possible, **locateNext** uses active indexes to speed the search. This method respects the limits of restricted views in linked detail tables.

The search begins with the record after the active record. If a match is found, the cursor moves to that record. This operation fails if the active record cannot be posted and unlocked (e.g., due to a key violation). If no match is found, the cursor returns to the active record. To start a search from the beginning of a table, use **locate**.

Note

- The search is case-sensitive unless **ignoreCaseInLocate** (Session type) is enabled.

Example

{button ,AL(`OPAL_TYPE_UIOBJECT;OPAL_METH_UILOCATE;OPAL_METH_UILOCATENEXTPATTERN;',0,"Defaultoverview",)} Related Topics

locateNext example

The following example assumes that a form contains a table frame bound to the *Customer* table and a button named *locateButton*. The **pushButton** method for *locateButton* searches for customers in the city of Freeport. If **locate** is successful, the code uses **locateNext** to find successive records.

```
; locateButton::pushButton
method pushButton(var eventInfo Event)
var
   Cust      UIObject
   searchFor String
   numFound  SmallInt
endVar
Cust.attach(CUSTOMER)
searchFor = "Freeport"
if Cust.locate("City", searchFor) then
   numFound = 1
   message("")
   while Cust.locateNext("City", searchFor)
      numFound = numFound + 1
   endWhile
   msgInfo("Found " + searchFor, strval(numFound) + " times.")
endIf

endmethod
```

locateNextPattern method

Locates the next record containing a field that has a specified pattern of characters.

Syntax

```
1. locateNextPattern ( [ const fieldName String,const exactMatch AnyType, ] * const fieldName String, const pattern String ) Logical

2. locateNextPattern ( [ const fieldNum SmallInt, const exactMatch AnyType, ] * const fieldNum SmallInt, const pattern String ) Logical
```

Description

locateNextPattern finds substrings (e.g., comp in computer). When possible, this method uses active indexes to speed the search. This method respects the limits of restricted views in linked detail tables.

The search begins with the record after the active record. If a match is found, the cursor moves to that record. This operation fails if the active record cannot be committed (e.g., due to a key violation). If no match is found, the cursor returns to the active record. To start a search from the beginning of a table, use **locatePattern**.

To search for records by the value of a single field, specify the field in *fieldName* or *fieldNum* (use *fieldNum* for faster performance) and specify a pattern of characters in *pattern*.

You can include the standard pattern operators **@** and **..** in the *pattern* argument. The **..** operator specifies any string of characters (including no string). The **@** operator specifies for any single character. Any combination of literal characters and wildcards can be used to construct a search. If **advancedWildCardsInLocate** (Session type) is enabled, you can use advanced match pattern operators. For more information, see the description of **advMatch**.

To search for records by the values of more than one field, specify exact matches on all fields except the last one in the list. For example, the following code searches the Name field for exact matches on the word Corel, the Product field for Corel Paradox, and the Keywords field for words beginning with data (e.g., database).

```
tc.locateNextPattern("Name", "Corel"  "Product", "Corel Paradox" "Keywords", "data..")
```

For examples, see Sample search strings with wildcards in the User's Guide help.

Note

- The search is case-sensitive unless **ignoreCaseInLocate** (Session type) is enabled.

Examples

{button ,AL(`OPAL_TYPE_UIOBJECT;OPAL_METH_UILOCATE;OPAL_METH_UILOCATEPATTERN;OPAL_METH_UILOCATENEXT;OPAL_METH_STAMAT;OPAL_METH_STMAT;',0,"Defaultoverview",)} Related Topics

locateNextPattern method examples

Example1 Simple searches

Example2 Searches based on multiple criteria

locateNextPattern example 1

The following example searches for multiple occurrences of the letter C in the Name field of the *Customer* table, and writes the matching names to an array. Assume that the *CUSTOMER* table frame is bound to *Customer*, and that *locateButton* is a button on the form.

```
; locateButton::pushButton
method pushButton(var eventInfo Event)
var
   Cust      UIObject           ; to attach to CUSTOMER table frame
   searchFor String             ; the pattern string to search for
   numFound  SmallInt           ; the number of matches located
   custNames Array[] String     ; the matches found
endVar


cust.attach(CUSTOMER)
searchFor = "C.."               ; find customers whose name
                                ; begins with C
if cust.locatePattern("Name", searchFor) then  ; if you can find one
  numFound = 1                                 ; post it to the array
  custNames.grow(1)                            ; then keep looking
  custNames[numFound] = cust."Name"
  while cust.locateNextPattern("Name", searchFor)
    numFound = numFound + 1
    custNames.grow(1)
    custNames[numFound] = cust."Name"
  endWhile
endIf
if custNames.size() > 0 then     ; if there's anything in the array
  custNames.view()               ; show the array
endIf
endMethod
```

locateNextPattern example 2

The following example searches for records by the value in the City field and the pattern in the Name field:

```
; locateButtonTwo::pushButton
method pushButton(var eventInfo Event)
var
   Cust      UIObject           ; to attach to CUSTOMER TableFrame
   searchFor String             ; the pattern string to search for
   numFound  SmallInt           ; the number of matches located
   custNames Array[] String     ; the matches found
endVar


cust.attach(CUSTOMER)
searchFor = "..C.."             ; find customers whose name
                                ; includes a C
if cust.locatePattern("City", "Marathon", "Name", searchFor) then  ; if you can find one
  numFound = 1                                  ; post it to the array
  custNames.grow(1)                             ; then keep looking
  custNames[numFound] = cust."Name"
  while cust.locateNextPattern("City", "Marathon", "Name", searchFor)
    numFound = numFound + 1
    custNames.grow(1)
    custNames[numFound] = cust."Name"
  endWhile
endIf
if custNames.size()  0 then     ; if there's anything in the array
  custNames.view()               ; show the array
endIf
endMethod
```

locatePattern method

Searches for a record containing a field that has a specified pattern of characters.

Syntax

```
1. locatePattern ( [ const fieldName String, const exactMatch AnyType, ] * const fieldName String, const pattern String ) Logical

2. locatePattern ( [ const fieldNum SmallInt, const exactMatch AnyType, ] * const fieldName SmallInt, const pattern String ) Logical
```

Description

locatePattern finds substrings (e.g., comp in computer). When possible, this method uses active indexes to speed the search. This method respects the limits of restricted views in linked detail tables.

The search begins with the record after the active record. If a match is found, the cursor moves to that record. This operation fails if the active record cannot be committed (e.g., due to a key violation). If no match is found, the cursor returns to the active record. To start a search from the beginning of a table, use **locatePattern**.

To search for records by the value of a single field, specify the field in *fieldName* or *fieldNum* (use *fieldNum* for faster performance) and specify a pattern of characters in *pattern*.

You can include the standard pattern operators **@** and **..** in the *pattern* argument. The **..** operator specifies any string of characters (including no string). The **@** operator specifies for any single character. Any combination of literal characters and wildcards can be used to construct a search. If **advancedWildCardsInLocate** (Session type) is enabled, you can use advanced match pattern operators. For more information, see the description of **advMatch**.

To search for records by the values of more than one field, specify exact matches on all fields except the last one in the list. For example, the following code searches the Name field for exact matches on the word Corel, the Product field for Corel Paradox, and the Keywords field for words beginning with data (e.g., database).

To start a search from the beginning of a table, use **locateNextPattern**.

```
tc.locateNextPattern("Name", "Corel"  "Product", "Corel Paradox" "Keywords", "data..")
```

For examples, see Sample search strings with wildcards in the User's Guide help.

Note

- The search is case-sensitive unless **ignoreCaseInLocate** (Session type) is enabled.

Example

{button ,AL(`OPAL_TYPE_UIOBJECT;OPAL_METH_UILOCATENEXTPATTERN;OPAL_METH_UILOCATE;OPAL_METH_STAMAT;OPAL_METH_STMAT;',0,"Defaultoverview",)} Related Topics

locatePattern example

See the **locateNextPattern** example.

locatePrior method

Searches backward from the active record for a specified field value.

Syntax

```
1. locatePrior ( const fieldName String, const exactMatch AnyType [ , const fieldName String,
const exactMatch AnyType ] * ) Logical

2. locatePrior ( const fieldNum SmallInt, const exactMatch AnyType [ , const fieldNum SmallInt,
const exactMatch AnyType ] * ) Logical
```

Description

locatePrior searches backwards from the active record in a table for record values that match one or more field/value pairs. Specify the search value in *exactMatch* and the search field in *fieldName* or *fieldNum* (use *fieldNum* for faster performance). When possible, **locateNext** uses active indexes to speed the search. This method respects the limits of restricted views in linked detail tables.

The search begins with the record before the active record and moves up through the table. If a match is found, the cursor moves to that record. This operation fails if the active record cannot be posted and unlocked (e.g., due to a key violation). If no match is found, the cursor returns to the active record. To start a search from the beginning of a table, use **locate**.

Note

- The search is case-sensitive unless **ignoreCaseInLocate** (Session type) is enabled.

Example

{button ,AL(`OPAL_TYPE_UIOBJECT;OPAL_METH_UILOCATE;OPAL_METH_UILOCATEPRIORPATTERN;',0,"Defaultoverview",)} Related Topics

locatePrior example

The following example locates the last occurrence of a value in a table by searching up from the end of the table using **locatePrior**. Assume that the form contains a table frame that is bound to the *Customer* table, and a button named *locateButton*.

```
; locateButton::pushButton
method pushButton(var eventInfo Event)
var
   Cust      UIObject      ; to attach to CUSTOMER table frame
   searchFor String        ; the string to search for
endVar
Cust.attach(CUSTOMER)      ; attach to table frame
Cust.end()                 ; move to the end of the table
searchFor = "Freeport"
if Cust.locatePrior("City", searchFor) then   ; find record
   msgInfo("Status", "The last record with a City of " +
           searchFor + " is record " + Cust.recno + ".")
endIf


endMethod
```

locatePriorPattern method

Searches backward from the active record for a field that contains a specified pattern of characters.

Syntax

```
1. locatePriorPattern ( [ const fieldName String, const exactMatch AnyType, ] * const fieldName String, const pattern String ) Logical

2. locatePriorPattern ( [ const fieldNum SmallInt, const exactMatch AnyType, ] * const fieldNum SmallInt, const pattern String ) Logical
```

Description

locatePriorPattern finds substrings (e.g., comp in computer). When possible, this method uses active indexes to speed the search. This method respects the limits of restricted views in linked detail tables.

The search begins with the record after the active record. If a match is found, the cursor moves to that record. This operation fails if the active record cannot be committed (e.g., due to a key violation). If no match is found, the cursor returns to the active record. To start a search at the beginning of a table, use **locatePattern**.

To search for records by the value of a single field, specify the field in *fieldName* or *fieldNum* (use *fieldNum* for faster performance) and specify a pattern of characters in *pattern*.

You can include the standard pattern operators **@** and **..** in the *pattern* argument. The **..** operator specifies any string of characters (including no string). The **@** operator specifies for any single character. Any combination of literal characters and wildcards can be used to construct a search. If **advancedWildCardsInLocate** (Session type) is enabled, you can use advanced match pattern operators. For more information, see the description of **advMatch**.

To search for records by the values of more than one field, specify exact matches on all fields except the last one in the list. For example, the following code searches the Name field for exact matches on the word Corel, the Product field for Corel Paradox, and the Keywords field for words beginning with data (e.g., database).

To start a search from the beginning of a table, use **locateNextPattern**.

```
tc.locateNextPattern("Name", "Corel"  "Product", "Corel Paradox" "Keywords", "data..")
```

For examples, see Sample search strings with wildcards in the User's Guide help.

Note

- The search is case-sensitive unless **ignoreCaseInLocate** (Session type) is enabled.

Example

{button ,AL(`OPAL_TYPE_UIOBJECT;OPAL_METH_UILOCATEPATTERN;OPAL_METH_UILOCATEPRIOR;OPAL_METH_STAMAT;OPAL_METH_STMAT;',0,"Defaultoverview",)} Related Topics

locatePriorPattern example

The following example locates the last occurrence of a value in a table by searching up from the end of the table and using **locatePriorPattern**. Assume that the form contains a table frame that is bound to the *Customer* table, and a button named *locateButton*.

```
; locateButton::pushButton
method pushButton(var eventInfo Event)
var
   Cust      UIObject      ; to attach to CUSTOMER table frame
   searchFor String        ; the string to search for
endVar
Cust.attach(CUSTOMER)      ; attach to table frame
Cust.end()                 ; move to the end of the table
searchFor = "Freeport"
if Cust.locatePrior("City", searchFor, "Name", "..C..") then   ; find record
   msgInfo("Status", "The last record with a City of " + searchFor +
           "and a name with C is record " + Cust.recno + ".")
endIf


endMethod
```

lockRecord method

Puts a write lock on the active record.

Syntax

lockRecord ( ) Logical

Description

lockRecord returns True if it places a write lock on the active record; otherwise, it returns False.

Note

- The Locked property is a read-only property. This means that you can't change the property setting to lock or unlock an object.

Examples

{button ,AL(`OPAL_TYPE_UIOBJECT;OPAL_METH_UIPOSTRECORD;OPAL_METH_UIRECORDSTATUS;OPAL_METH_UIUNLOCKRECORD;',0,"Defaultoverview",)} Related Topics

lockRecord method examples

Example1 Locking a record

Example2 Determining whether a record is locked

lockRecord example 1

The following example determines whether the *Customer* table is in Edit mode. If it is, the method locates a record, attempts to lock it using **lockRecord** and determines the status of the lock using **recordStatus**. Assume that a form contains a table frame that is bound to the *Customer* table, and a button named *lockButton*. The record inside the *CUSTOMER* table frame is named *custRec*.

```
; lockButton::pushButton
method pushButton(var eventInfo Event)
var
  obj  UIObject
endVar
obj.attach(CUSTOMER)
obj.locate("Name", "Sight Diver")
if thisForm.editing then
  if CUSTOMER.isEdit() then
    if NOT obj.lockRecord() then
       msgStop("Lock failed", "recordStatus(\"Locked\") is " +
             String(obj.recordStatus("Locked")))
    else
       msgStop("Lock succeeded", "recordStatus(\"Locked\") is " +
             String(obj.recordStatus("Locked")))
       obj.custRec."Name" = "Right Diver"  ; quotes on Name indicate
                                           ; field name instead of
                                           ; property
       obj.unlockRecord()
    endIf
  else
    msgInfo("Status", "You must be in edit mode to lock and change records.")
  endIf

endIf
endMethod
```

lockRecord example 2

The following example examines a record object's Locked property:

```
; lockButtonTwo::pushButton
method pushButton(var eventInfo Event)
var
  obj,
  recObj  UIObject
endVar


obj.attach(CUSTOMER)
obj.locate("Name", "Sight Diver")


if thisForm.editing then
  obj.lockRecord()                ; no write access to Locked property
                                  ; so use method to lock record
  recObj.attach(CUSTOMER.custRec)
  if NOT recObj.Locked then       ; check the property to see
                                  ; if the record is locked
     msgStop("Lock failed", "recObj.Locked is " +
             String(recObj.Locked))
  else
     msgStop("Lock succeeded", "recObj.Locked is " +
             String(recObj.Locked))
     recObj."Name" = "Right Diver" ; name is in quotes to indicate Name
                                  ; field instead of obj's Name property
     obj.unlockRecord()
  endIf
else
  msgInfo("Status", "You must be in edit mode to lock and change records.")
endIf
endMethod
```

lockStatus method

Returns the number of locks on a table.

Syntax

```
lockStatus (const lockType String ) SmallInt
```

Description

lockStatus returns the number of locks of type *lockType* on a table. *lockType*'s value is Write, Read, or Any.

If you haven't placed any locks on the table **lockStatus** returns 0.

If you specify Any for *lockType*, **lockStatus** returns the total number of locks you've placed on the table. **lockStatus** does not include locks placed by Corel Paradox or by other users or applications.

Example

{button ,AL(`OPAL_TYPE_UIOBJECT;OPAL_METH_UIENUMLOCKS;OPAL_METH_UILOCKRECORD;',0,"Default overview",)} Related Topics

lockStatus example

The following example assumes that a form has a table frame named *CUSTOMER* that is bound to the *Customer* table, and a button named *lockButton*. The **pushButton** method for *lockButton* removes all locks from *CUSTOMER*, searches for locks using **lockStatus**, places a lock and reports on the locks using **lockStatus**.

```
; lockButton::pushButton
method pushButton(var eventInfo Event)
var
  CustTC TCursor     ; to place a lock on the table
  Cust UIObject
  l    Logical
endVar
CustTC.attach(CUSTOMER)     ; attach the TCursor to CUSTOMER
l = unlock(CustTC, "ALL")   ; remove any locks
l.view("Unlock successful:")
Cust.attach(CUSTOMER)       ; attach the UIObject to CUSTOMER
if Cust.lockStatus("ANY") = 0 then   ; check for locks
   l = lock(CustTC, "WL")            ; place a write lock
   l.view("Lock successful:")        ; check up on it
endIf
msgInfo("Status", "Table " + Cust.Name + " has " +
         String(Cust.lockStatus("WL")) + " write lock(s).")
unlock(CustTC, "ALL")               ; remove any locks
endMethod
```

menuAction method/procedure

Sends an event to an object's **menuAction** method.

Syntax

```
menuAction ( const action SmallInt )
```

Description

menuAction constructs a MenuEvent and sends it to a specified UIObject's **menuAction** method. *action* is one of the MenuCommands constants, or a user-defined menu constant.

Note

- You can't use menuAction to simulate a File menu command. To send a menu command constant that is equivalent to a File menu command, use one of the regular Action constants, manipulate a property, or use a System type method.

Example

{button ,AL(`OPAL_TYPE_UIOBJECT;OPAL_METH_UIACTI;',0,"Defaultoverview",)} Related Topics

menuAction example

The following example uses the *sendATile* button on the current form to send *thisForm* a MenuWindowTile action.

```
; sendATile::pushButton
method pushButton(var eventInfo Event)
thisForm.menuAction(MenuWindowTile)
endMethod
```

methodDelete method

Deletes a specified method.

Syntax

```
methodDelete ( const methodName String ) Logical
```

Description

methodDelete deletes the method specified by *methodName*. The form that contains the object must be in a Form Design window.

Example

{button ,AL(`OPAL_TYPE_UIOBJECT;OPAL_METH_UICREATE;OPAL_METH_UIMETHODGET;OPAL_METH_UIMETHODSET;',0,"Defaultoverview",)} Related Topics

methodDelete example

The following example uses **methodGet**, **methodSet**, and **methodDelete** to copy methods from one object to another. The code overrides the **pushButton** method for a button named *copyMethods*. The form contains four other objects. The *targetForm* field lets you specify the name of the form containing the objects to copy. The *sourceObject* field holds the name of the object containing the methods to copy. The *destinationObject* field contains the name of the object to copy the methods to. The final object is a radio button field named *copyOrMove* which specifies whether methods in the source are copied, or copied then deleted.

```
; copyMethods::pushButton
method pushButton(var eventInfo Event)
var
  otherForm            Form       ; a handle to a form
  sourceObj,                      ; object to copy from
  destObj              UIObject   ; object to copy to
  methodStr            String     ; stores the method definition
  methodArray Array[]  String     ; holds method names to copy
  i                    SmallInt   ; array index
endvar


; open the form and attach to the objects
if targetForm = "" OR sourceObject = "" OR destinationObject = "" then
  msgStop("Error", "Please fill in form, source, and destination.")
  return
endIf
if NOT otherForm.load(targetForm.value) then
  msgStop("Error", "Couldn't open named form.")
  return
endIf
if NOT sourceObj.attach(otherForm, sourceObject.value) then
  otherForm.close()
  msgStop("Error", "Couldn't find source object. Please specify entire path.")
  return
endIf
if NOT destObj.attach(otherForm, destinationObject.value) then
  otherForm.close()
  msgStop("Error", "Couldn't find destination object. Specify entire path.")
  return
endIf


; set up the array of method names to copy
methodArray.addLast("mouseUp")
methodArray.addLast("mouseDown")
methodArray.addLast("mouseDouble")
methodArray.addLast("mouseEnter")
methodArray.addLast("mouseExit")
methodArray.addLast("mouseRightUp")
methodArray.addLast("mouseRightDown")
methodArray.addLast("mouseRightDouble")
methodArray.addLast("mouseMove")
methodArray.addLast("open")
methodArray.addLast("close")
methodArray.addLast("canArrive")
methodArray.addLast("arrive")
methodArray.addLast("setFocus")
methodArray.addLast("canDepart")
methodArray.addLast("depart")
methodArray.addLast("removeFocus")
methodArray.addLast("depart")
methodArray.addLast("timer")
methodArray.addLast("keyPhysical")
methodArray.addLast("keyChar")
methodArray.addLast("action")
methodArray.addLast("menuAction")
methodArray.addLast("error")
methodArray.addLast("status")


; add the method names specific to fields and buttons
```

```
if sourceObj.class = "Field" AND destObj.class = "Field" then
  methodArray.addLast("changeValue")
  methodArray.addLast("newValue")
else
if sourceObj.class = "Button" AND destObj.class = "Button" then
  methodArray.addLast("pushButton")
endIf
if sourceObj.class <> "Button" AND destObj.class <> "Button" then
  methodArray.addLast("mouseClick")
endIf


; copy methods from sourceObj to destObj on form otherForm
for i from 1 to methodArray.size()
  ; write the method named in methodArray to the string
;  msgInfo("methodArray is", methodArray[i])
  try
    methodStr = sourceObj.methodGet(methodArray[i])
    msgInfo("FYI", "Retrieved " + methodArray[i] + " method.")
    ; write the string to the method named in methodArray
    destObj.methodSet(methodArray[i], methodStr)
    if copyOrMove.Value = "Move" then
      sourceObj.methodDelete(methodArray[i])
    endIf
  onfail
  ;  loop
  endTry
endfor


endMethod
```

methodEdit method

Opens an object's method in an Editor window.

Syntax

```
methodEdit (const methodName String) Logical
```

Description

methodEdit opens the method specified by *methodName* in an Editor window. If you specify a method that doesn't exist, **methodEdit** will create it for you. **methodEdit** fails if you try to open a method that is running.

Example

{button ,AL(`OPAL_TYPE_UIOBJECT;OPAL_METH_UIMETHODGET;OPAL_METH_UIMETHODSET;OPAL_METH_UIMETHODDELETE;;',0,"Defaultoverview",)} Related Topics

methodEdit example

The following example opens the object's **testMethod** method in an Editor window:

```
method pushButton(var eventInfo Event)
var
    MyForm form
    MyObject  uiobject
endvar

MyForm.load("vendors.fsl")
MyObject.attach(MyForm,"Preferred")

MyObject.methodEdit("testMethod")
endMethod
```

methodGet method

Returns the text of a specified method.

Syntax

```
methodGet ( const methodName String ) String
```

Description

methodGet returns the text of the method specified in *methodName*.

Example

{button ,AL(`OPAL_TYPE_UIOBJECT;OPAL_METH_UICREATE;OPAL_METH_UIMETHODDELETE;OPAL_METH_UIMETHODSET;',0,"Defaultoverview",)} Related Topics

methodGet example

See the **methodDelete** example.

methodSet method

Sets the text of a specified method.

Syntax

```
methodSet ( const methodName String, const methodText String ) Logical
```

Description

methodSet specifies the source code for the method named in *methodName*. Open the form that contains the object in a Form Design window.

Example

{button ,AL(`OPAL_TYPE_UIOBJECT;OPAL_METH_UICREATE;OPAL_METH_UIMETHODDELETE;OPAL_METH_UIMETHODGET;',0,"Defaultoverview",)} Related Topics

methodSet example

See the **methodDelete** example.

mouseClick method

Sends an event to an object's **mouseClick** method.

Syntax

```
mouseClick ( ) Logical
```

Description

mouseClick constructs a **mouseClick** MouseEvent to call the object's built-in **mouseClick** event method.

Example

{button ,AL(`OPAL_TYPE_UIOBJECT;OPAL_METH_UIMUP;',0,"Defaultoverview",)} Related Topics

mouseClick example

The following example sends a **mouseClick** MouseEvent to *fieldTwo* on the form:

```
; sendMouseClick::pushButton
method pushButton(var eventInfo Event)
; send a mouseClick to fieldTwo
fieldTwo.mouseClick()
endMethod
```

mouseDouble method

Sends an event to an object's **mouseDouble** method.

Syntax

```
mouseDouble ( const x LongInt, const y LongInt, const state SmallInt ) Logical
```

Description

mouseDouble constructs a double-click event to call the object's built-in **mouseClick** event method. The event's coordinates are specified in *x* and *y* (in twips). Specify the mouse and keyboard state in *state* using KeyboardStates constants. You can add these constants together to create combined key states (e.g., CTRL + Left Arrow key).

Example

{button ,AL(`OPAL_TYPE_UIOBJECT;OPAL_METH_UIMRDOUB;OPAL_METH_UIMDOWN;OPAL_METH_UIMUP;',0,"Defaultoverview",)} **Related Topics**

mouseDouble example

The following example sends a double-click to *fieldTwo* on the form:

```
; sendMouseDouble::pushButton
method pushButton(var eventInfo Event)
; send a mouseDouble to fieldTwo
fieldTwo.mouseDouble(100, 100, LeftButton)
endMethod
```

mouseDown method

Sends an event to an object's **mouseDown** method.

Syntax

```
mouseDown ( const x LongInt, const y LongInt, const state SmallInt ) Logical
```

Description

mouseDown constructs an event to call the object's built-in **mouseDown** event method. The event's coordinates are specified in x and y (in twips). Specify the mouse and keyboard state in *state* using KeyboardStates constants. You can add these constants together to create combined key states (e.g., Left Arrow key + CTRL).

Example

{button ,AL(`OPAL_TYPE_UIOBJECT;OPAL_METH_UIMUP;OPAL_METH_UIMDOUB;OPAL_METH_UIMRDOWN;OPAL_METH_UIMRUO;',0,"Defaultoverview",)} Related Topics

mouseDown example

The following example sends a **mouseDown** and a **mouseUp** MouseEvent to the object *fieldOne* on the form:

```
method pushButton(var eventInfo Event)
var
  fPt  Point
endVar
fPt = fieldOne.Position
fieldOne.mouseDown(fPt.x(), fPt.y(), LeftButton)
sleep(500)
fieldOne.mouseUp(fPt.x(), fPt.y(), LeftButton)
endMethod
```

mouseEnter method

Sends an event to an object's **mouseEnter** method.

Syntax

```
mouseEnter ( const x LongInt, const y LongInt, const state SmallInt ) Logical
```

Description

mouseEnter constructs an event to call the object's built-in **mouseEnter** event method. The event's coordinates are specified in x and y (in twips). Specify the mouse and keyboard state in *state* using KeyboardStates constants. You can add these constants together to create combined key states (e.g., Left Arrow key + CTRL).

Example

**{button ,AL(`OPAL_TYPE_UIOBJECT;OPAL_METH_UIMEXIT;OPAL_METH_UIMMOVE;',0,"Defaultoverview",)
} Related Topics**

mouseEnter example

The following example sends a **mouseEnter** MouseEvent to a field named *fieldSix* on the form:

```
; sendMouseEnter::pushButton
method pushButton(var eventInfo Event)
; send a mouseEnter to fieldSix
fieldSix.mouseEnter(100,100,LeftButton)
endMethod
```

mouseExit method

Sends an event to an object's **mouseExit** method.

Syntax

```
mouseExit ( const x LongInt, const y LongInt, const state SmallInt ) Logical
```

Description

mouseExit constructs an event to call the object's built-in **mouseExit** event method. The event's coordinates are specified in x and y (in twips). Specify the mouse and keyboard state in *state* using KeyboardStates constants. You can add these constants together to create combined key states (e.g., Left Arrow key + CTRL).

Example

{button ,AL(`OPAL_TYPE_UIOBJECT;OPAL_METH_UIMENTER;OPAL_METH_UIMMOVE;',0,"Defaultoverview",)} Related Topics

mouseExit example

The following example sends a **mouseExit** MouseEvent to *fieldSeven* on the form:

```
; sendMouseExit::pushButton
method pushButton(var eventInfo Event)
; send a mouseExit to fieldSeven
fieldSeven.mouseExit(100, 100, LeftButton)
endMethod
```

mouseMove method

Sends an event to an object's **mouseMove** method.

Syntax

```
mouseMove ( const x LongInt, const y LongInt, const state SmallInt ) Logical
```

Description

mouseMove constructs an event to call the object's built-in **mouseMove** event method. The event's coordinates are specified in x and y (in twips). Specify the mouse and keyboard state in *state* using KeyboardStates constants. You can add these constants together to create combined key states (e.g., Left Arrow key + CTRL).

Example

**{button ,AL(`OPAL_TYPE_UIOBJECT;OPAL_METH_UIMENTER;OPAL_METH_UIMEXIT;',0,"Defaultoverview",)
} Related Topics**

mouseMove example

The following example sends a **mouseDown**, a **mouseUp**, and a **mouseMove** MouseEvent to a field named *fieldFive* on the form:

```
; sendMouseMove::pushButton
method pushButton(var eventInfo Event)
fieldFive.mouseDown(100, 100, LeftButton)
fieldFive.mouseUp(100, 100, LeftButton)
; send a mouseMove to fieldFive
fieldFive.mouseMove(100, 100, LeftButton)
endMethod
```

mouseRightDouble method

Sends an event to an object's **mouseRightDouble** method.

Syntax

```
mouseRightDouble ( const x LongInt, const y LongInt, const state SmallInt ) Logical
```

Description

mouseRightDouble constructs an event to call the object's built-in **mouseRightDouble** event method. The event's coordinates are specified in x and y (in twips). Specify the mouse and keyboard state in *state* using KeyboardStates constants. You can add these constants together to create combined key states (e.g., Left Arrow key + CTRL).

Example

{button ,AL(`OPAL_TYPE_UIOBJECT;OPAL_METH_UIMDOUB;OPAL_METH_UIMRUO;OPAL_METH_UIMRDOWN;',0,"Defaultoverview",)} Related Topics

mouseRightDouble example

The following example sends a **mouseRightDouble** MouseEvent to a field named *fieldTwo* on the form:

```
; sendMouseDouble::pushButton
method pushButton(var eventInfo Event)
; send a mouseDouble to fieldTwo
fieldTwo.mouseDouble(100, 100, LeftButton)
endMethod
```

mouseRightDown method

Sends an event to an object's **mouseRightDown** method.

Syntax

```
mouseRightDown ( const x LongInt, const y LongInt, const state SmallInt ) Logical
```

Description

mouseRightDown constructs an event to call the object's built-in **mouseRightDown** event method. The event's coordinates are specified in x and y (in twips). Specify the mouse and keyboard state in *state* using KeyboardStates constants. You can add these constants together to create combined key states (e.g., Left Arrow key + CTRL).

Example

{button ,AL(`OPAL_TYPE_UIOBJECT;OPAL_METH_UIMRUO;OPAL_METH_UIMRDOUB;OPAL_METH_UIMUP;OPAL_METH_UIMDOWN;',0,"Defaultoverview",)} Related Topics

mouseRightDown example

The following example sends a **mouseRightDown** and a **mouseRightUp** MouseEvent to a field named *fieldThree* on the form:

```
; sendMouseRightUp::pushButton
method pushButton(var eventInfo Event)
var
  fPt Point
endVar
fP = fieldThree.position    ; get the position, send a mouseRightDown
fieldThree.mouseRightDown(fPt.x(), fPt.y(), LeftButton)
sleep(500)                  ; pause and send a mouseRightUp
fieldThree.mouseRightUp(fPt.x(), fPt.y(), LeftButton)
endMethod
```

mouseRightUp method

Sends an event to an object's **mouseRightUp** method.

Syntax

```
mouseRightUp ( const x LongInt, const y LongInt, const state SmallInt ) Logical
```

Description

mouseRightUp constructs an event to call the object's built-in **mouseRightUp** event method. The event's coordinates are specified in x and y (in twips). Specify the mouse and keyboard state in *state* using KeyboardStates constants. You can add these constants together to create combined key states (e.g., Left Arrow key + CTRL).

Example

{button ,AL(`OPAL_TYPE_UIOBJECT;OPAL_METH_UIMRDOWN;OPAL_METH_UIMRDOUB;OPAL_METH_UIMDOWN;OPAL_METH_UIMUP;',0,"Defaultoverview",)} Related Topics

mouseRightUp example

The following example sends a **mouseRightDown** and a **mouseRightUp** MouseEvent to a field named *fieldThree* on the form:

```
; sendMouseRightUp::pushButton
method pushButton(var eventInfo Event)
var
  fPt Point
endVar
fP = fieldThree.position    ; get the position, send a mouseRightDown
fieldThree.mouseRightDown(fPt.x(), fPt.y(), LeftButton)
sleep(500)                  ; pause and send a mouseRightUp
fieldThree.mouseRightUp(fPt.x(), fPt.y(), LeftButton)
endMethod
```

mouseUp method

Sends an event to an object's **mouseUp** method.

Syntax

```
mouseUp ( const x LongInt, const y LongInt, const state SmallInt ) Logical
```

Description

mouseUp constructs an event to call the object's built-in **mouseUp** event method. The event's coordinates are specified in x and y (in twips). Specify the mouse and keyboard state in *state* using KeyboardStates constants. You can add these constants together to create combined key states (e.g., Left Arrow key + CTRL).

Example

{button ,AL(`OPAL_TYPE_UIOBJECT;OPAL_METH_UIMDOWN;OPAL_METH_UIMDOUB;OPAL_METH_UIMRDOWN;OPAL_METH_UIMRUO;OPAL_METH_UIMRDOUB;',0,"Defaultoverview",)} Related Topics

mouseUp example

The following example sends a **mouseDown** and a **mouseUp** MouseEvent o the object *fieldOne* on the form:

```
method pushButton(var eventInfo Event)
var
  fPt  Point
endVar
fPt = fieldOne.Position
fieldOne.mouseDown(fPt.x(), fPt.y(), LeftButton)
sleep(500)
fieldOne.mouseUp(fPt.x(), fPt.y(), LeftButton)
endMethod
```

moveTo method

Sets the focus to a specified object.

Syntax

```
1. (Method) moveTo ( ) Logical
2. (Procedure) moveTo ( const objectName String ) Logical
```

Description

moveTo moves the focus to a specified object. When you call **moveTo** as a procedure (Syntax 2), *objectName* specifies the destination object (the object to which the focus is moved).

Example

{button ,AL(`OPAL_TYPE_UIOBJECT;OPAL_METH_UIATTA;',0,"Defaultoverview",)} Related Topics

moveTo example

The following example assumes that a form contains a table frame that is bound to *Orders*, and another table frame that is bound to *LineItem*. *Orders* has a one-to-many link to *LineItem*. The form also contains a button named *findDetails* . In this example, the **pushButton** method for *findDetails* searches the entire table for orders that include the current part number.

The following code is attached to the Var window for *findDetails*:

```
; findDetails::Var
Var
  lineTC TCursor  ; instance of LINEITEM for searching
endVar


; findDetails::open
method open(var eventInfo Event)
lineTC.open("LineItem.db")
endMethod
```

The following code is attached to *findDetails*' **pushButton** method:

```
; findDetails::pushButton
method pushButton(var eventInfo Event)
var
  stockNum  Number
  orderTC    TCursor
  OrderNum    Number
endVar


; get Stock No from current LineItem
stockNum = LINEITEM.lineRecord."Stock No"
; lineTC was declared in Var window and opened by open method
if NOT lineTC.locateNext("Stock No", stockNum) then
  lineTC.locate("Stock No", stockNum)
endIf
orderTC.attach(ORDERS)
orderTC.locate("Order No", lineTC."Order No")
ORDERS.moveToRecord(orderTC)       ; move to CUSTOMER and
                                   ; resynchronize with TCursor
LINEITEM.lineRecord."Stock No".moveTo()   ; move cursor to LINEITEM detail
; move cursor to matching record
LINEITEM.locate("Stock No", stockNum)
endMethod
```

The following code is attached to *findDetails*' **close** method:

```
; findDetails::close
method close(var eventInfo Event)
lineTC.close()   ; close the TCursor to LineItem
endMethod
```

moveToRecNo method

Moves to a specific record in a dBASE table.

Syntax

```
moveToRecNo ( const recordNum LongInt ) Logical
```

Description

moveToRecNo sets the active record to *recordNum*. This method returns an error if *recordNum* is not in the table. Use **nRecords** or examine the NRecords property to determine the number of records in a table. Use **moveToRecNo** only for dBASE tables. Use **moveToRecord** for Corel Paradox tables.

Example

{button ,AL(`OPAL_TYPE_UIOBJECT;OPAL_METH_UINREC;OPAL_METH_UIMOTOREC;OPAL_METH_UIRESYNC;',0,"Defaultoverview",)} Related Topics

moveToRecNo example

The following example moves to the middle record in a table. Assume that a form contains a table frame that is bound to the *LineItem* table, and a button named *MidWay*.

```
; MidWay::pushButton
method pushButton(var eventInfo Event)
var
  halfWay  LongInt
endVar


halfWay = LongInt(LINEITEM.nRecords()/2)
LINEITEM.moveToRecNo(halfWay)


endMethod
```

moveToRecord method

Moves to a specific record in a table.

Syntax

```
1. moveToRecord ( const recordNum LongInt ) Logical
2. moveToRecord ( const tc TCursor ) Logical
```

Description

moveToRecord moves to a specific record in a table.

Syntax 1 moves to the record number specified in *recordNum*. This method returns an error if *recordNum* is greater than the number of records in the table. Use the method **nRecords** or examine the NRecords property to determine the number of records in a table.

Syntax 2 moves to the record pointed to by the TCursor *tc*. Use **moveToRecNo** to accelerate performance in dBASE tables.

Example

{button ,AL(`OPAL_TYPE_UIOBJECT;OPAL_METH_UINREC;OPAL_METH_UIMOTO;OPAL_METH_UIMOVETORECNO;OPAL_METH_UIRESYNC;',0,"Defaultoverview",)} Related Topics

moveToRecord example

The following example moves to the middle record of a table. Assume that the form contains a table frame that is bound to the *LineItem* table, and a button named *MidWay*. For an example of how to use **moveToRecord** using a TCursor, see the **moveTo** example.

```
; MidWay::pushButton
method pushButton(var eventInfo Event)
var
  halfWay  LongInt
endVar


halfWay = LongInt(LINEITEM.nRecords()/2)
LINEITEM.moveToRecord(halfWay)


endMethod
```

nextRecord method

Moves to the next record in a table.

Syntax

```
nextRecord ( ) Logical
```

Description

nextRecord moves to the next record in a table. This method returns an error if the cursor is already at the last record.

nextRecord has the same effect as the action constant DataNextRecord. This means that the following statements are equivalent:

```
obj.nextRecord()
obj.action(DataNextRecord)
```

Example

{button ,AL(`OPAL_TYPE_UIOBJECT;OPAL_METH_UIHOME;OPAL_METH_UIEND;OPAL_METH_UIPRIO;OPAL_METH_UIMOTOREC;',0,"Defaultoverview",)} Related Topics

nextRecord example

The following example moves to the next record in the *Customer* table. Assume that *Customer* is bound to a table frame on the form and that *moveToNext* is a button on the form.

```
; moveToNext::pushButton
method pushButton(var eventInfo Event)
if NOT CUSTOMER.atLast() then
  CUSTOMER.nextRecord()  ; move to the next record
  ; same as:  CUSTOMER.action(DataNextRecord)
  msgInfo("What record?", CUSTOMER.recno)
else
  msgInfo("Status", "Already at the last record.")
endIf
endMethod
```

nFields method

Returns the number of fields in a table.

Syntax

```
nFields ( ) LongInt
```

Description

nFields returns the number of fields in a table. To determine the number of columns displayed in an object that is bound to a table, examine the value of the NCols property for that object.

Example

{button ,AL(`OPAL_TYPE_UIOBJECT;OPAL_METH_UINKEY;OPAL_METH_UINREC;',0,"Defaultoverview",)}

Related Topics

nFields example

The following example returns the number of fields and key fields in the *LineItem* table. Assume that a form has a table frame named *LINEITEM* that is bound to the *LineItem* table, and a button named *tableStats*.

```
; tableStats::pushButton
method pushButton(var eventInfo Event)
msgInfo("Status", "The LineItem table has " +
        String(LINEITEM.nFields()) + " fields and " +
        String(LINEITEM.nKeyFields()) + " key fields." +
        "\nThere are " + String(LINEITEM.NCols) +
        " columns in the table frame.")
endMethod
```

nKeyFields method

Returns the fields in the active index.

Syntax

```
nKeyFields ( ) LongInt
```

Description

nKeyFields returns the number of fields in the index associated with a UIObject.

Example

{button ,AL(`OPAL_TYPE_UIOBJECT;OPAL_METH_UINFIE;OPAL_METH_UINREC;',0,"Defaultoverview",)}

Related Topics

nKeyFields example

See the **nFields** example.

nRecords method

Returns the number of records in a table.

Syntax

nRecords ( ) LongInt

Description

nRecords returns the number of records in a table that is bound to a table frame, multi-record object, or field object. You can also examine an object's NRecords property to determine the number of records in the table bound to that object. Both operations are time consuming for dBASE tables and large Corel Paradox tables.

The **nRecords** method and the NRecords property respect the limits of restricted views. If a table-based object is the detail table in a one-to-many relationship, **nRecords** reports the number of linked detail records, not the total number of records in the table.

For a Corel Paradox table, **nRecords** returns the number of records in the underlying table, not the number of records displayed in the object. For example, if the *Customer* table contains 100 records and a table frame that is bound to the *Customer* table displays 5 records, this method would return 100, not 5.

For a dBASE table, **nRecords** counts deleted records if they are displayed in the form. To make a form display deleted records, choose Form, Show Deleted, or call **action(DataShowDeleted)** or **action(DataToggleDeleted)**.

Note

- When you call nRecords after setting a filter, the returned value does not represent the number of records in the filtered set. To retrieve the number of records in the filtered set, attach a TCursor to the UIObject and call **cCount**.When you call nRecords after setting a range, the returned value represents the number of records in the set that are defined by the range.

Example

{button ,AL(`OPAL_TYPE_UIOBJECT;OPAL_METH_UIMOTOREC;OPAL_METH_UINKEY;OPAL_METH_UINFIE;',0,"Defaultoverview",)} Related Topics

nRecords example

The following example moves to the middle record in a table. Assume that a form contains a table frame named *LINEITEM* that is bound to the *LineItem* table, and a button named *MidWay*.

```
; MidWay::pushButton
method pushButton(var eventInfo Event)
var
  halfWay  LongInt
endVar


halfWay = LongInt(LINEITEM.nRecords()/2)
LINEITEM.moveToRecord(halfWay)


endMethod
```

pixelsToTwips method

Converts screen coordinates from pixels to twips.

Syntax

```
pixelsToTwips ( const pixels Point ) Point
```

Description

pixelsToTwips converts the screen coordinates from pixels to twips. A pixel is a dot on the screen, and a twip is a unit equal to 1/1440 of a logical inch (1/20 of a printer's point).

Example

{button ,AL(`OPAL_TYPE_UIOBJECT;OPAL_METH_UITWIPSTOPIXELS;',0,"Defaultoverview",)} Related Topics

pixelsToTwips example

The following example assumes that a form contains a two-inch square box named *twoSquare*. The *twoSquare* box contains two text boxes: *pixNum* and *twipNum. pixNum* displays the width of the box in pixels and *twipNum* displays the width of the box in twips.

```
; twoSquare::mouseUp
method mouseUp(var eventInfo MouseEvent)
var
  twTopLeft,             ; top left point in twips
  twBottomRight,         ; bottom right point in twips
  pxTopLeft,             ; top left in pixels
  pxBottomRight,         ; bottom right in pixels
  selfPos      Point     ; current position property
endvar
self.getBoundingBox(twTopLeft, twBottomRight)    ; returns points in twips
twipNum.Text = twBottomRight.x() - twTopLeft.x() ; get the width in twips
pxTopLeft = TwipsToPixels(twTopLeft)             ; convert to pixels
pxBottomRight = TwipsToPixels(twBottomRight)
pixNum.Text = pxBottomRight.x() - pxTopLeft.x()  ; get the width in pixels
; cross check
twTopLeft = PixelsToTwips(pxTopLeft)      ; convert from pixels back to twips
twTopLeft.view("Top left in twips")       ; twTopLeft should match selfPos
selfPos = self.Position                   ; get selfPos, twips by default
selfPos.view("Position of box in twips")  ; show the result
endMethod
```

postAction method

Posts an action to an action queue for delayed execution.

Syntax

```
postAction ( const actionId SmallInt )
```

Description

postAction posts an action to an action queue for delayed execution. This method works like **action**, except that the action is not executed immediately. Instead, the action is posted to an action queue when the method is called. Corel Paradox waits until a yield occurs (e.g., the current method completes execution or calls **sleep**).

The value of *actionID* can be a user-defined action constant or a constant from one of the following Action classes:

ActionDataCommands
ActionEditCommands
ActionFieldCommands
ActionMoveCommands
ActionSelectCommands

Example

{button ,AL(`OPAL_TYPE_UIOBJECT;OPAL_METH_UIACTI;OPAL_METH_FOPOSTACTION;OPAL_METH_UIACTI;OPAL_METH_UIBROADCASTACTION;OPAL_METH_UIMENUACTION;OPAL_TYPE_ACTIONEVENT;OPAL_ACTIONEVENT_USERDEFINEDCONSTANTS;',0,"Defaultoverview",)} Related Topics

postAction example

The following example demonstrates how to store a value from a calculated field in a table. In this example, an unbound calculated field object named *fldLineTotal* calculates the line total. Whenever the calculation occurs, **postAction** sends a custom user action. This custom user action posts the value to a table frame that is bound to the *Lineitem* table.

The following code defines the calculation for the calculated field.

```
;fldLineTotal :: Calculation
[LINEITEM.SELLING PRICE]*[LINEITEM.QTY]   ;Calculated field.
```

The following code is attached to the field object's built-in **newValue** method.

```
;fldLineTotal :: newValue
method newValue(var eventInfo Event)
   if Qty.isEdit() then            ;If edit mode,
      Qty.postAction(UserAction + 1)      ;send a custom user
   endIf                         ;action to QTY.
endmethod
```

The following code is attached to the table frame's built-in **action** method.

```
;recTFrame :: action
method action(var eventInfo ActionEvent)
   if eventInfo.id() = UserAction + 1 then   ;If ID is user
      dmPut("LINEITEM", "Total", Total.value)   ;action and
      Qty.postRecord()               ;post changes.
   endIf
endmethod
```

postRecord method

Posts a pending record to a table.

Syntax

```
postRecord ( ) Logical
```

Description

postRecord returns True if the active record is successfully posted to the underlying table; otherwise, it returns False. **postRecord** does not unlock a locked record.

postRecord has the same effect as the action constant DataPostRecord. This means that the following statements are equivalent:

```
obj.postRecord()
obj.action(DataPostRecord)
```

Example

{button ,AL(`OPAL_TYPE_UIOBJECT;OPAL_METH_UIRECORDSTATUS;OPAL_METH_UILOCKRECORD;OPAL_METH_TCATTACHTOKEYVIOL;',0,"Defaultoverview",)} Related Topics

postRecord example

The following example locates a record, uses **lockRecord** to lock it, and determines the status of the lock using **recordStatus**. The code changes the record and posts it using **postRecord**. Assume that a form contains a table frame that is bound to the *Customer* table, and a button named *lockButton*.

```
; lockButton::pushButton
method pushButton(var eventInfo Event)
var
  obj  UIObject
endVar
obj.attach(CUSTOMER)
obj.locate("Name", "Sight Diver")
if thisForm.Editing then
  if NOT obj.lockRecord() then
     msgStop("Lock failed", "recordStatus(\"Locked\") is " +
             String(obj.recordStatus("Locked")))
  else
     msgStop("Lock succeeded", "recordStatus(\"Locked\") is " +
             String(obj.recordStatus("Locked")))
     obj.custRec."Name" = "Right Diver"  ; quotes on Name indicates
                                         ; field name instead of property
     obj.postRecord()
     message("Record is locked: ", obj.custRec.locked)
  endIf
else
  msgInfo("Status", "You must be in edit mode to lock and change records.")
endIf
endMethod
```

priorRecord method

Moves to the previous record in a table.

Syntax

priorRecord ( ) Logical

Description

priorRecord moves to the previous record in a table. This method returns an error if the cursor is already at the first record.

priorRecord has the same effect as the action constant DataPriorRecord. This means that the following statements are equivalent:

```
obj.priorRecord()
obj.action(DataPriorRecord)
```

Example

{button ,AL(`OPAL_TYPE_UIOBJECT;OPAL_METH_UIHOME;OPAL_METH_UIEND;OPAL_METH_UINEXTR;OPAL_METH_UICURR;OPAL_METH_UISKIP;OPAL_METH_UIMOTOREC;',0,"Defaultoverview",)} Related Topics

priorRecord example

The following example moves to the previous record in the *Customer* table. Assume that *Customer* is bound to a table frame on the form and that *moveToPrior* is a button on the form.

```
; moveToPrior::pushButton
method pushButton(var eventInfo Event)
if NOT CUSTOMER.atFirst() then
  CUSTOMER.priorRecord()  ; move to the previous record
  ; same as CUSTOMER.action(DataPriorRecord)
  msgInfo("What record?", CUSTOMER.recno)
else
  msgInfo("Status", "Already at the first record.")
endIf
endMethod
```

pushButton method

Generates a **pushButton** event and sends it to an object.

Syntax

pushButton ( ) Logical

Description

pushButton creates a **pushButton** event to call the object's built-in **pushButton** method of an object with that event.

Example

{button ,AL(`OPAL_TYPE_UIOBJECT;OPAL_METH_UIMOUSECLICK;OPAL_METH_UIMUP;',0,"Defaultoverview",)} **Related Topics**

pushButton example

The following example sends a **pushButton** event to *buttonTwo* on the form:

```
; sendPushButton::pushButton
method pushButton(var eventInfo Event)
; send a pushButton to buttonTwo
buttonTwo.pushButton()
endMethod
```

recordStatus method

Reports the status of a record.

Syntax

```
recordStatus ( const statusType String ) Logical
```

Description

recordStatus returns True or False answers to a question about the status of a record. Use the argument *statusType* to specify the status in question (i.e., is New, Locked, or Modified).

The New value means the record has just been added to the table. Locked means that an implicit or explicit lock has been placed on the record. Modified means at least one of the field values has been changed. You can also obtain information about the active record by examining the Inserting, Locked, Focus, and Touched properties for the record.

 **Example**

{button ,AL(`OPAL_TYPE_UIOBJECT;OPAL_METH_UILOCKRECORD;OPAL_METH_UIUNLOCKRECORD;',0,"Defaultoverview",)} Related Topics

recordStatus example

The following example locates a record, attempts to lock it using **lockRecord** and determines the status of the lock using **recordStatus**. The method changes the record and unlocks it using **unlockRecord**. Assume that a form contains a table frame that is bound to the *Customer* table and a button named *lockButton*. The record object of the table frame is named *custRec*.

```
; lockButton::pushButton
method pushButton(var eventInfo Event)
var
  Cust   UIObject
  newKey Number
endVar
Cust.attach(CUSTOMER)               ; attach to CUSTOMER table frame
Cust.locate("Name", "Sight Diver")  ; find the record
if NOT isEdit() then        ; check if form is in Edit mode
  msgInfo("Status", "You must be in Edit mode for this operation.")
else
  if NOT Cust.lockRecord() then     ; try to lock the record
     msgStop("Status", "Lock Failed. recordStatus(\"Locked\") is " +
             String(Cust.recordStatus("Locked")))
  else
     msgInfo("Record locked?", Cust.recordStatus("Locked"))
     newKey = 1384
     Cust.custRec.Customer_No.value = newKey   ; change the key value
     Cust.custRec.Customer_No.action(EditCommitField)
     msgInfo("Record modified?", Cust.recordStatus("Modified"))
     Cust.unlockRecord()            ; try to unlock the recordif it
                                    ; causes a keyviol, Corel Paradox
                                    ; leaves record locked
      if Cust.recordStatus("Locked") then
       msgInfo("Status", "Record was a key violation. Changing key.")
       newKey = 1451
       Cust.custRec.Customer_No.value = newKey ; change to a new key
       Cust.postRecord()                  ; post it
       ; record will "fly away" to a new position based on key
     endIf
     Cust.locate("Customer No", newKey)   ; find the "fly away"
  endIf
endIf
endMethod
```

resync method

Resynchronizes an object to a TCursor.

Syntax

```
resync ( const tc TCursor ) Logical
```

Description

resync changes the active record pointer of a UIObject to the active record of a TCursor named *tc*. When you resynchronize a table object to a TCursor, the table's filters and indexes are changed to those of the TCursor. A dBASE table also takes the Show Deleted setting of the TCursor.

Note

- **resync** only applies when the UIObject and the TCursor are associated with the same table.

Example

{button ,AL(`OPAL_TYPE_UIOBJECT;OPAL_METH_UIATTA;OPAL_METH_UIMOTOREC;OPAL_METH_UIIREC;opal_meth_uiibef;',0,"Defaultoverview",)} Related Topics

resync example

See the **insertBeforeRecord** example.

rgb method

Defines a color.

Syntax

```
rgb ( const red SmallInt, const green SmallInt, const blue SmallInt ) LongInt
```

Description

rgb defines a color using *red*, *green*, and *blue*, which can be integers ranging from 0 to 255, or Colors constants.

Example

{button ,AL(`OPAL_TYPE_UIOBJECT;OPAL_METH_UIGPRO;OPAL_METH_UIGETRGB;OPAL_METH_UISPRO;',0,"Defaultoverview",)} Related Topics

rgb example

The following example uses **rgb** to set the color of boxes as they're created. The code also creates a color palette. Assume that the titles exist on the form in the appropriate locations. The form has a button named *showPalette*.

```
; drawPalette::pushButton
method pushButton(var eventInfo Event)
var
  palAr Array[5] SmallInt   ; array to hold rgb values
  setBaseX       LongInt    ; base position
  setBaseY       LongInt    ; base position
  ui             UIObject   ; handle to create boxes
endVar
const
  horizInc = 720            ; amount to move horizontally (twips)
  vertInc =  540            ; amount to move vertically
endConst


palAr[1] = 0
palAr[2] = 64
palAr[3] = 128
palAr[4] = 192
palAr[5] = 255


for i from 1 to palAR.size()        ; reds(diagonal position)
  setBaseX = 720 + ((i - 1) * 150)  ; change base as i increases
  setBaseY = 720 + ((i - 1) * 150)
  for j from 1 to palAR.size()      ; greens (vertical positioning)
    for k from 1 to palAR.size()    ; blue   (horizontal positioning)
      ui.create(boxTool, setBaseX + (horizInc * (k - 1)),
                setBaseY + (vertInc * (j - 1)), 250, 250)
      ; set the color using rgb and values from array
      ui.Color = rgb(palAr[i], palAr[j], palAr[k])
      ui.Visible = Yes
    endfor                          ; k (blue, horizontal)
  endfor                            ; j (green, vertical)
endfor                              ; i (red, diagonal)


endMethod
```

sendToBack method

Displays an object behind other objects.

Syntax

```
sendToBack ( )
```

Description

sendToBack moves a UIObject to a window's back drawing layer, displaying it behind other objects. If the UIObject is a form, this method displays the form window behind other windows. **sendToBack** works in design mode and run mode and you do not have to select the object. Use **sendToBack** if

- you have objects that overlap each other
- you want to rearrange the tab order

When you change the position of an object, you also change its tab order. An object always tabs from back to front.

Example

{button ,AL(`OPAL_TYPE_UIOBJECT;OPAL_METH_UIBRINGTOFRONT;',0,"Defaultoverview",)} Related Topics

sendToBack example

The following example assumes that a form contains two multi-record objects that occupy the same location and size. Two buttons toggle between each multi-record object: *btnShowVendors* and *btnShowStock*. *btnShowVendors* uses **sendToBack** to send the *STOCK* multi-record object to the background; the *VENDORS* multi-record object is in front. *btnShowStock* uses **sendToBack** to send the *VENDORS* multi-record object to the background; the *STOCK* multi-record object is in front.

The following code is attached to *btnShowVendors*.

```
;btnShowVendors :: pushButton
method pushButton(var eventInfo Event)
   STOCK.sendToBack()   ; Send the VENDORS MRO to the back
   Vendor_No.moveTo()   ; so the STOCK MRO may be seen.
endmethod
```

The following code is attached to *btnShowStock*.

```
;btnShowStock :: pushButton
method pushButton(var eventInfo Event)
   VENDORS.sendToBack()   ; Send the STOCK MRO to the back
   Stock_No.moveTo()   ; so the VENDORS MRO may be seen.
endmethod
```

setGenFilter method

Specifies conditions for including records in a field, table frame, or multi-record object.

Syntax

```
1. setGenFilter ( [ idxName String, [ tagName String, ] ] criteria DynArray [ ] AnyType )
Logical

2. setGenFilter ( [ idxName String, [ tagName String, ] ] criteria Array[ ] AnyType [ , fieldId
Array[ ] AnyType ] ) Logical
```

Description

setGenFilter specifies conditions for including records in a field, table frame, or multi-record object. Records that meet the specified conditions are included, and all remaining records are filtered out. Unlike **setRange**, this method does not require an indexed table. **setGenFilter** must be executed before opening a table using a TCursor.

In Syntax 1, a dynamic array (DynArray) named *criteria* specifies the index as the field name or number, and the item as the criteria expression. For example, the following code specifies criteria based on the values of three fields:

```
; The value of the first field in the table is Widget.

criteriaDA[1]      = "Widget"
; The value of the field named Size is greater than 4.

criteriaDA["Size"] = "> 4"
; The value of the field named Cost is greater than or equal to 10.95
; and less than 22.50.

criteriaDA["Cost"] = ">= 10.95, < 22.50"
```

If the DynArray is empty or contains at least one empty item, any existing filter criteria are removed.

In Syntax 2, the array named *criteria* specifies conditions, and the optional Array *fieldId* specifies field names and numbers. If you omit *fieldID*, conditions are applied to fields in the order that they appear in the *criteria* array (the first condition applies to the first field in the table, the second condition applies to the second field, etc.). The following example fills arrays for Syntax 2 to specify the criteria outlined in the Syntax 1 example.

```
criteriaAR[1] = "Widget"
criteriaAR[2] = "> 4"
criteriaAR[3] = ">= 10.95, < 22.50"
fieldAR[1] = 1
fieldAR[2] = "Size"
fieldAR[3] = "Cost"
```

If the Array is empty or contains at least one empty item, the existing filter criteria is removed.

For both syntaxes, *idxName* specifies an index name (Corel Paradox and dBASE tables) and *tagName* specifies a tag name (dBASE tables only). If you use these optional items, the index and tag are applied to the underlying table before the filtering criteria.

This method fails if the active record cannot be committed.

Note

- If you use **setGenFilter** on a UIObject in a running report, the filter does not take effect until you run the report again. For example, the following code runs a report and sets a filter; however, the filter has no effect until the report switches to design mode and then back into run mode.

```
method pushButton(var eventInfo Event)
   var
      reOrders     Report
      daCriteria   DynArray[] AnyType
   endVar


   reOrders.open("orders")


   daCriteria["OrderNo"] = "> 1234"


; Assume the report contains a table frame bound to the Orders table.
; This statement has no effect because the report is in run mode.
   reOrders.ORDERS.setGenFilter(daCriteria)
```

```
    reOrders.design()
    reOrders.run() ; Now the filter takes effect.
endMethod
```

Example

{button ,AL(`OPAL_TYPE_UIOBJECT;OPAL_METH_UIGETGENFILTER;OPAL_METH_UIDROPGENFILTER;OPAL_METH_UISETRANGE;OPAL_METH_TCSETGENFILTER;OPAL_INFO_TBUSINGRANGESANDFILTERS;',0,"Default overview",)} Related Topics

setGenFilter example

The following example uses the **pushButton** method for a button named *balanceDueBtn* uses **setGenFilter** to filter a table frame on a form. This code filters the *ORDERS* table frame to display only those orders with a positive balance due.

```
;balanceDueBtn :: pushButton
method pushButton(var eventInfo Event)
   var
      dyn               DynArray[] String
      stField, stData   String
   endVar


   stField = "Balance Due"
   stData = "> 0"
   dyn[stField] = stData


   ORDERS.setGenFilter(dyn)   ; ORDERS is a detail table frame.
endmethod
```

setPosition method

Sets the position of an object.

Syntax

```
setPosition ( const x LongInt, const y LongInt, const w LongInt, const h LongInt)
```

Description

setPosition sets the position of an object on the screen. Variables *x* and *y* specify the coordinates of an object's upper-left corner (in twips). Variables *w* and *h* specify the object's width and height (in twips). If the object is not specified, self is implied.

This method does not work when the UIObjects are forms. To set the position of a form, use **setPosition** (Form type).

You can also set and examine an object's position and size using the Position and Size properties.

```
self.Position = Point(100, 150)
self.Size = Point(2000, 2500)
```

The following code performs that same function as the previous code:

```
self.setPosition(100, 150, 2000, 2500)
```

For ObjectPAL, the screen is a two-dimensional grid. The origin (0, 0) is located at the upper-left corner of an object's container, with positive x values extending to the right, and positive y values extending down.

For dialog boxes and for the Corel Paradox desktop application, the position is specified relative to the entire screen . For forms, reports, and table windows, the position is specified relative to the Corel Paradox desktop.

Example

{button ,AL(`OPAL_TYPE_UIOBJECT;OPAL_METH_UIGETPOSITION;OPAL_METH_FOSPOS;',0,"Defaultoverview",)} Related Topics

setPosition example

The following example moves a circle across the screen in response to timer events. The **pushButton** method for *toggleButton* uses **setTimer** and **killTimer** to start or stop a timer, respectively, depending on the condition of the button. When the timer starts, it issues a timer event every 100 milliseconds. Each timer event causes *toggleButton's* **timer** method to execute. The **timer** method retrieves the current position of the ellipse using **getPosition** and moves it 100 twips to the right using **setPosition**.

The following code is attached to *toggleButton*'s **pushButton** method:

```
; toggleButton::pushButton
method pushButton(var eventInfo Event)
; label for button was renamed to buttonLabel
if buttonLabel = "Start Timer" then   ; if stopped, then start
  buttonLabel = "Stop Timer"          ; change label
  self.setTimer(10)                   ; start the timer
else                                  ; if started, then stop
  buttonLabel = "Start Timer"         ; change label
  self.killTimer()                    ; stop the timer
endIf


endMethod
```

The following code is attached to *toggleButton*'s **timer** method:

```
; toggleButton::timer
method timer(var eventInfo TimerEvent)
var
  ui          UIObject
  x, y, w, h  SmallInt
endVar
ui.attach(floatCircle)         ; attach to the circle
ui.getPosition(x, y, w, h)     ; assign coordinates to vars
if x  4320 then                ; if not at left edge of area
  ui.setPosition(x + 100, y, w, h)  ; move to the left
else
  ui.setPosition(1440, y, w, h)     ; return to the right
endIf
endMethod
```

setProperty method

Sets a property to a specified value.

Syntax

```
setProperty ( const propertyName String, const propertyValue AnyType )
```

Description

setProperty sets an object's *propertyName* property to *propertyValue*. If the object does not have a *propertyName* property, or if *propertyValue* is invalid, this method fails.

setProperty is especially useful when *propertyName* is a variable; otherwise, you can access the property directly using the following code:

```
aBox.Color = Red
```

Example

**{button ,AL(`OPAL_TYPE_UIOBJECT;OPAL_METH_UIGPRO;OPAL_METH_UIGPROST;',0,"Defaultoverview",)
} Related Topics**

setProperty example

The following example creates a dynamic array that's indexed by property names and contains property values. The array is filled using the array's index as the argument to the **getProperty** command. The method changes one of the property values and resets the object's properties using the **setProperty** method.

```
; boxOne::mouseUp
method mouseUp(var eventInfo MouseEvent)
var
  propNames DynArray[] AnyType    ; to hold property names & values
  arrayIndex           String     ; index to dynamic array
endVar


propNames["Color"] = ""
propNames["Visible"] = ""
propNames["Name"] = ""


foreach arrayIndex in propNames
  propNames[arrayIndex] = self.getProperty(arrayIndex)
endforeach


propNames["Color"] = "DarkBlue"


foreach arrayIndex in propNames
  self.setProperty(arrayIndex, propNames[arrayIndex])
endforeach


endMethod
```

setRange method

Specifies a range of records to include in a field, table frame, or multi-record object. This method replaces **setFilter** that was included in earlier versions of Corel Paradox. Code that calls **setFilter** executes as before.

Syntax

```
1. setRange ( [ const exactMatchVal AnyType] * [ , const minVal AnyType, const maxVal AnyType ] ) Logical
2. setRange ( rangeVals Array[ ] AnyType ) Logical
```

Description

setRange specifies a range of records to include in a field, table frame, or multi-record object. **setRange** compares the criteria that you specify with values in the corresponding fields of a table's index. If the active record cannot be committed or if the table is not indexed, this method fails. If you call **setRange** without any arguments the range criteria is reset to include the entire table.

In Syntax 1, you must specify values in *minVal* and *maxVal* to set a range based on the value of the first field of the index. For example, the following code determines values in the first field of each record's index:

```
tblObj.setRange(14, 88)
```

If a value is less than 14 or greater than 88, its corresponding record is excluded from the range. To specify an exact match on the first field of the index, assign *minVal* and *maxVal* the same value. For example, the following code excludes all values except 55:

```
tblObj.setRange(55, 55)
```

To set a range based on the values of more than one field specify exact matches *except* for the last one in the list. For example, the following statement looks for exact matches on Corel and Corel Paradox, and on values ranging from 100 to 500, inclusive, for the third field:

```
tblObj.setRange("Corel", "Corel Paradox", 100, 500)
```

In Syntax 2, you can pass an array of values to specify the range criteria. The following table displays the number of array items and their corresponding range specifications:

Number of array items	Range specification
No items (empty array)	No items resets range criteria to include the entire table.
One item	One item specifies a value for an exact match on the index's first field.
Two items	Two items specifies a range for the index's first field.
Three items	The first item specifies an exact match for the index's first field; items 2 and 3 specify a range for the index's second field.
More than three items	For an array of size *n*, specify exact matches on the index's first *n*-2 fields. The last two array items specify a range for the index's *n*-1 field.

Examples

setRange method examples

Example1 Setting simple ranges

Example2 Setting ranges based on three or more criteria

setRange example 1

For the following example, assume that the first field in *Lineitem*'s key is Order No. and you want to know the total for order number 1005. When you press the *getDetailSum* button, the **pushButton** method limits the number of records included in the LINEITEM object, including only those with 1005 in the first key field.

```
; getDetails::pushButton
method pushButton(var eventInfo Event)
var
  tblObj  UIObject
endVar
if tblObj.attach(LINEITEM) then


  ; this limits tblObj's view to records that have
  ; 1005 as their key value (Order No. 1005).
  tblObj.setRange(1005, 1005)
  ; now display the number of records for Order No. 1005
  msgInfo("Total records for order 1005", tblObj.nRecords())
else
  msgStop("Sorry", "Can't attach to table.")
endIf
endMethod
```

setRange example 2

The following example calls **setRange** with a criteria array that contains more than three items. The following code instructs a table frame to display orders from a person with a specific first name, middle initial, and last name. This table frame displays only those orders that range from 100 to 500 items. This example assumes that the *PartsOrd* table is indexed on the FirstName, MiddleInitial, LastName, and Qty fields.

```
; setQtyRange::pushButton
method pushButton(var eventInfo Event)
   var
      arRangeInfo   Array[5] AnyType
   endVar


   arRangeInfo[1] = "Frank"       ; FirstName (exact match)
   arRangeInfo[2] = "P."          ; MiddleInitial (exact match)
   arRangeInfo[3] = "Corel"     ; LastName (exact match)
   arRangeInfo[4] = 100           ; Minimum qty value
   arRangeInfo[5] = 500           ; Maximum qty value


   PartsOrd.setRange(arRangeInfo) ; PartsOrd is a table frame
endMethod
```

setTimer method

Starts an object's timer.

Syntax

```
setTimer ( const milliSeconds LongInt [ , const repeat Logical ] )
```

Description

setTimer starts an object's timer. The timer interval (in milliseconds) is specified using *milliSeconds*. The optional argument *repeat* specifies whether the timer automatically repeats. If *repeat* is set to True or omitted, the timer repeats; otherwise, the timer event is sent once. **setTimer** is attached to an object's **open** method, and the object's response is defined in its **timer** method.

Note

- Although Windows allows a maximum of 16 timers for all applications, Corel Paradox has no timer limit.

Example

{button ,AL(`OPAL_TYPE_UIOBJECT;OPAL_METH_UIKITI;',0,"Defaultoverview",)} Related Topics

setTimer example

The following example moves a circle across the screen in response to timer events. The **pushButton** method for *toggleButton* uses **setTimer** and **killTimer** to start or stop a timer, respectively, depending on the condition of the button. When the timer starts, it issues a timer event every 100 milliseconds. Each timer event causes *toggleButton's* **timer** method to execute. The **timer** method retrieves the ellipse's position using **getPosition** and moves it 100 twips to the right using **setPosition**.

The following code is for *toggleButton*'s **pushButton** method:

```
; toggleButton::pushButton
method pushButton(var eventInfo Event)
if buttonLabel = "Start Timer" then   ; if stopped, then start
  buttonLabel = "Stop Timer"          ; change label
  self.setTimer(10)                   ; start the timer
else
  buttonLabel = "Start Timer"         ; change label
  self.killTimer()                    ; stop the timer
endIf


endMethod
```

The following code is for *toggleButton*'s **timer** method:

```
; toggleButton::timer
method timer(var eventInfo TimerEvent)
var
  ui          UIObject
  x, y, w, h  SmallInt
endVar
ui.attach(floatCircle)         ; attach to the circle
ui.getPosition(x, y, w, h)     ; assign coordinates to vars
if x  4320 then                ; if not at left edge of area
  ui.setPosition(x + 100, y, w, h)  ; move to the left
else
  ui.setPosition(1440, y, w, h)     ; return to the right
endIf
endMethod
```

skip method

Moves forward or backward through a specified number of records.

Syntax

```
skip ( const nRecords LongInt ) Logical
```

Description

skip moves forward or backward through a specified number of records. If you attempt to move beyond the limits of the table, **skip** fails.

Specifying a positive value for *nRecords* moves forward through the table, specifying a negative value moves backward, and setting *nRecords* to 0 leaves the table as it is.

Note

- Setting *nRecords* = 0 is the same as **currRecord**
- Setting *nRecords* = -1 is the same as **priorRecord**
- Setting *nRecords* = 1 is the same as **nextRecord**

Example

{button ,AL(`OPAL_TYPE_UIOBJECT;OPAL_METH_UIHOME;OPAL_METH_UIEND;OPAL_METH_UINEXTR;OPAL_METH_UIPRIO;OPAL_METH_UICURR;OPAL_METH_UIMOTOREC;',0,"Defaultoverview",)} Related Topics

skip example

The following example fills a table with records from the *Orders* table. Assume that the table *SampOrd* already exists with the same structure as *Orders*. The *createSampling* button exists on a form along with a table frame that is bound to *Orders. CreateSampling's* **pushButton** method is shown below. The code moves the cursor through the *Orders* table, skips a random number of records, and copies the record it lands on to the sampling table.

```
; createSampling::pushButton
method pushButton(var eventInfo Event)
var
  ordSampleTC         TCursor       ; handle to sampling table
  copyRec Array[]     String        ; holds record copied from Orders
  randInt             SmallInt      ; random number to skip
  OrdObj              UIObject      ; handle to Orders
endVar


ordObj.attach(ORDERS)               ; attach to ORDERS table frame
ordObj.home()                       ; move to the first record
if ordSampleTC.open("OrdSamp.db")  then
  ordSampleTC.empty()               ; clear out sampling table
  ordSampleTC.edit()                ; start editing
  while NOT OrdObj.atLast()
    randInt = int(rand() * 20) + 1  ; create an integer between 1 and 20
    randInt.view()                  ; show the number
    OrdObj.skip(randInt)            ; skip a random number of records
    OrdObj.copyToArray(copyRec)     ; get the record
    ordSampleTC.insertRecord()      ; make a space for it
    ordSampleTC.copyFromArray(copyRec)  ; insert the record
  endwhile
  ordSampleTC.endEdit()             ; end editing
  msgInfo("Status", "OrdSamp table now has " +
           String(ordSampleTC.nRecords()) + " records.")
  ordSampleTC.close()               ; close it out
else
  msgStop("Oops", "Sorry. Couldn't find OrdSamp table.")
endIf
endMethod
```

switchIndex method

Specifies another index to use for viewing a table's records.

Syntax

1. switchIndex ([const ***indexName*** String] [**,** const ***stayOnRecord*** Logical] **)** Logical

2. switchIndex ([const ***indexFileName*** String] [**,** const ***tagName*** String [**,** const ***stayOnRecord*** Logical]] **)** Logical

Description

switchIndex specifies an index file to use with a table. In Syntax 1, *indexName* specifies an index to use with a Corel Paradox table. If you omit *indexName*, the table's primary index is used.

Syntax 2 is for dBASE tables. *indexFileName* can specify an .NDX file or an .MDX file, and optional argument *tagName* specifies an index tag in a production index file (.MDX).

In both syntaxes, if optional argument *stayOnRecord* is set to Yes, this method maintains the active record after the index switch. If it is set to No, the first record in the table becomes the active record. If omitted, *stayOnRecord* is set to No by default.

For more information on indexes, see About keys and indexes in tables in the User's Guide help.

Example

{button ,AL(`OPAL_TYPE_UIOBJECT;OPAL_METH_UISETGENFILTER;OPAL_METH_UISETRANGE;OPAL_METH_TCREIND;OPAL_METH_TCREINAL;OPAL_METH_TBSIND;',0,"Defaultoverview",)} Related Topics

switchIndex example

The following example assumes that *Customer* is a keyed Corel Paradox table that has a secondary index named NameAndState. This example attaches to a table frame bound to *Customer*, and calls **switchIndex** to switch from the primary index to the NameAndState index.

```
; thisButton::pushButton
method pushButton(var eventInfo Event)
var
  tblObj UIObject
endvar


tblObj.attach(CUSTOMER)               ; attach to Customer
tblObj.switchindex("NameAndState")    ; switch to index NameAndState
tblObj.home()                         ; make sure we're on the first record
msgInfo("First Record", tblObj."Name")  ; display value in Name field
; quotes around "Name" distinguish field name from property name
endMethod
```

twipsToPixels method

Converts screen coordinates from twips to pixels.

Syntax

```
twipsToPixels ( const twips Point ) Point
```

Description

twipsToPixels converts the screen coordinates specified in *twips* from twips to pixels. A pixel is a dot on the screen, and a twip is a device-independent unit equal to 1/1440 of a logical inch (1/20 of a printer's point).

Example

{button ,AL(`OPAL_TYPE_UIOBJECT;OPAL_METH_UIPIXELSTOTWIPS;',0,"Defaultoverview",)} Related Topics

twipsToPixels example

See the **pixelsToTwips** example.

unDeleteRecord method

Restores the current record in a dBASE table.

Syntax

unDeleteRecord ( ) Logical

Description

unDeleteRecord restores the current record of a dBASE table. This operation is successful if **showDeleted** has been set to True, if the record is deleted, and if the table object is in Edit mode.

Example

{button ,AL(`OPAL_TYPE_UIOBJECT;OPAL_METH_UIDELE;OPAL_METH_UIRECORDDELETED;OPAL_METH_TCISSHOWDELETEDON;OPAL_METH_TCSHDEL;',0,"Defaultoverview",)} Related Topics

unDeleteRecord example

See the **unDeleteRecord** (Tcursor type) example.

unlockRecord method

Removes a write lock from the active record.

Syntax

unlockRecord ( ) Logical

Description

unlockRecord returns True if it successfully removes an explicit write lock on the active record; otherwise, it returns False.

Note

- The Locked property is a read-only property. You can determine whether an object is locked, but you cannot lock or unlock an object.

Example

{button ,AL(`OPAL_TYPE_UIOBJECT;OPAL_METH_UIRECORDSTATUS;OPAL_METH_UILOCKRECORD;OPAL_METH_TCATTACHTOKEYVIOL;OPAL_METH_TCDIDFLYAWAY;OPAL_METH_TCSETFLYAWAYCONTROL;',0,"Defaultoverview",)} Related Topics

unlockRecord example

See the **recordStatus** example.

view method

Displays the value of an object in a dialog box.

Syntax

```
view ( [ const title String ] )
```

Description

view displays the value of an object in a dialog box. Corel Paradox suspends method execution until you close the dialog box. You can specify, in *title*, a title for the dialog box in the title string. If you omit *title*, the dialog box's title becomes the value's data type.

This method works only with the following UIObjects:

- buttons as checkboxes or buttons
- unbound fields only as lists or buttons
- fields bound to a table (the field's data type can be any data type except Memo and Graphic)

Calling **view** with any other UIObject causes a run-time error.

Example

{button ,AL(`OPAL_TYPE_UIOBJECT;OPAL_METH_UIATTA;',0,"Defaultoverview",)} **Related Topics**

view example

The following example assumes that a form contains a table frame named *CUSTOMER* that is bound to the *Customer* table, and a button. The following code is attached to the button's **pushButton** method. This code creates an array of seven UIObjects and views each item in the array.

```
;  page::mouseUp
method mouseUp(var eventInfo MouseEvent)
var
   obj          UIObject
   arr Array[7] UIObject
   i            SmallInt
endVar
arr[1].attach(CUSTOMER.Phone) ; the Phone field (A15) in the table frame
                              ; shows the phone number
arr[2].attach(aGraphic)       ; a bitmap (invalid)
arr[3].attach(someText)       ; a text object (invalid)
arr[4].attach(someList)       ; an unbound list field
                              ; shows the list item selected
arr[5].attach(someUnField)    ; an unbound field (invalid)
arr[6].attach(someRadio)      ; an unbound field as a radio button
                              ; shows the value of the active radio button
arr[7].attach(someButton)     ; an unbound field as a checkbox
                              ; True if checked, otherwise False
for i from 1 to arr.size()
  arr[i].view(arr[1].Class + ": Item " + String(i))
endFor
endMethod
```

wasLastClicked method

Determines whether an object received the last mouse click.

Syntax

```
wasLastClicked ( ) Logical
```

Description

wasLastClicked returns True if an object received the last mouse click; otherwise, it returns False. This method is only used with objects in the active form.

Example

{button ,AL(`OPAL_TYPE_UIOBJECT;OPAL_METH_UIWLRC;OPAL_METH_UIHAMO;',0,"Defaultoverview",)}

Related Topics

wasLastClicked example

The following example attaches code to the **mouseUp** method for an object named *boxOne*. If *boxOne* received the click, the message appears. If *boxOne* was sent a **mouseUp** event from another object, the method beeps.

The following code is attached to *boxOne*'s **mouseUp** method:

```
; boxOne::mouseUp
method mouseUp(var eventInfo MouseEvent)
if self.wasLastClicked() then
  msgInfo("Hey!", "Quit clicking me.")  ; method invoked by clicking
else
  beep()                                ; method invoked indirectly
endIf
endMethod
```

The following code is attached to *sendAClick*'s **mouseUp** method:

```
; sendAClick::mouseUp
method mouseUp(var eventInfo MouseEvent)
boxOne.mouseUp(eventInfo)  ; when boxOne's mouseUp gets this,
                           ; it will beep
endMethod
```

wasLastRightClicked method

Determines whether an object received the last right-mouse click.

Syntax

wasLastRightClicked ( ) Logical

Description

wasLastRightClicked returns True if an object received the last right-mouse click; otherwise, it returns False. This method is only used with objects in the active form.

Example

{button ,AL(`OPAL_TYPE_UIOBJECT;OPAL_METH_UIWLCL;OPAL_METH_UIHAMO;',0,"Defaultoverview",)}

Related Topics

wasLastRightClicked example

The following example is attached to the **mouseRightUp** method for an object named *circleOne*. If the ellipse received a right-click, the specified message displays. If the ellipse was sent a **mouseRightUp** event from another object, the code displays an alternate message.

The following code is attached to *circleOne*'s **mouseRightUp** method:

```
; circleOne::mouseRightUp
method mouseRightUp(var eventInfo MouseEvent)
if self.wasLastRightClicked() then
  ; method invoked by right-click
  msgInfo("Right-click", "Go click on someone your own size.")
else
  msginfo("Sent Right-click", "Invoked indirectly") ; method invoked indirectly
endIf
endMethod
```

The following is attached to the **mouseRightUp** method for an object named *sendARightClick*. When this object receives a right-click, it will send the event to *circleOne*.

The following code is attached to *sendARightClick*'*s* **mouseRightUp** method:

```
;sendARightClick::MouseRightUp

Method mouseRightUp(var eventInfo MouseEvent)

circleOne.mouseRightUp(eventInfo) ; when circleOne

;gets this it will trigger the second message box

endMethod
```

ValueEvent type

ValueEvent methods control field value changes. In fact, the **changeValue** built-in event method is the only method triggered by a ValueEvent. This means that the built-in **newValue** method is not called with a ValueEvent; instead, **newValue** takes an Event.

The built-in **changeValue** method is called when the a field value is about to change. **changeValue** allows you to determine whether you want to post the value. The built-in **newValue** method reports when a field has already received a new value. Fields defined as buttons or lists behave differently. The built-in **newValue** method differs from the **newValue** method for the ValueEvent type.

The ValueEvent type includes several derived methods from the Event type.

Methods for the ValueEvent type

Event		**ValueEvent**
errorCode		**newValue**
getTarget		**setNewValue**
isFirstTime		
isPreFilter		
isTargetSelf		
reason		
setErrorCod		
setReason		

Print related ObjectPAL methods and examples

newValue method

Returns a new, unposted value for a ValueEvent.

Syntax

```
newValue ( ) AnyType
```

Description

newValue returns the new, unposted value for a ValueEvent. Because the new value is not yet assigned to a field, the following two statements might return different values:

```
field.Value
eventInfo.newValue()
```

Note

- **newValue** differs from the built-in **newValue** method.

Example

{button ,AL(`OPAL_TYPE_VALUEEVENT;OPAL_METH_VESNVAL;',0,"Defaultoverview",)} Related Topics

newValue example

In the following example, the **changeValue** method for the creditLimit field compares the old value with the new value. If the difference between the old and new values is greater than 25 per cent, **changeValue** blocks the change. Assume that *creditLimit* is an unbound field on a form, and that the form has at least one other field.

```
; creditLimit::changeValue
method changeValue(var eventInfo ValueEvent)
var
  oldVal,
  newVal   Number
endVar
oldVal = self.Value              ; the property may be different
newVal = eventInfo.newValue()    ; than the new value
if (newVal > oldVal) AND (oldVal <> 0) then
  if (newVal - oldVal)/oldVal > 0.25 then
    msgStop("Stop", "You are not allowed to increase the " +
                    "credit limit more than 25%.")
    self.action(EditUndoField)  ; use this to restore old value
    eventInfo.setErrorCode(CanNotDepart)   ; block departure
  endIf
endIf
endMethod
```

setNewValue method

Specifies a value to set for a ValueEvent.

Syntax

```
setNewValue ( const newValue AnyType )
```

Description

setNewValue specifies a value to set for a ValueEvent. Ensure that the data type of the value is consistent with the field's type.

Example

{button ,AL(`OPAL_TYPE_VALUEEVENT;OPAL_METH_VENVAL;',0,"Defaultoverview",)} Related Topics

setNewValue example

The following example assumes that a form contains a field named *authorAbbrToName*, and at least one other field. When the user types an author abbreviation and moves off the field, **changeValue** fills in the full author name.

```
; authorAbbrToName::changeValue
method changeValue(var eventInfo ValueEvent)
var
  abbrValue,
  fullValue String
endVar


abbrValue = upper(eventInfo.newValue())  ; get the value and convert
                                         ; to uppercase
; user enters an abbreviation--change to full name
switch
  case abbrValue = "AC" : fullValue = "Agatha Christie"
  case abbrValue = "SP" : fullValue = "Sara Paretsky"
  case abbrValue = "MHC": fullValue = "Mary Higgins Clark"
  case abbrValue = "FK" : fullValue = "Faye Kellerman"
  case abbrValue = "SG" : fullValue = "Susan Grafton"
  case abbrValue = "AF" : fullValue = "Antonia Fraser"
  otherwise : fullValue = "Author Unknown"
endswitch

eventInfo.setNewValue(fullValue)
endMethod
```

Alphabetical list of ObjectPAL methods

B

C

cAverage (Table Type)
cAverage (TCursor Type)
cCount (Table Type)
cCount (TCursor Type)
ceil
charAnsiCode
char
checkField
checkRow
chrOEM
chr
chrToKeyName
clearCheck
clipboardErase
clipboardHasFormat
close (AddinForm Type)
close (Database Type)
close (DDE Type)
close (Form Type)
close (Library Type)
close (OleAuto Type)
close (Report Type)
close (Session Type)
close (System Type)
close (TCursor Type)
close (TextStream Type)
close (TableView Type)
closeQuery
cMax (Table Type)
cMax (TCursor Type)
cMin (Table Type)
cMin (TCursor Type)
cNpv (Table Type)
cNpv (TCursor Type)
commit
commitTransaction
compact (Table Type)
compact (TCursor Type)
compileInformation
constantNameToValue
constantValueToName
contains (Array Type)
contains (DynArray Type)
contains (Menu Type)
convertPointWithRespectTo
copyFromArray (TCursor Type)
copyFromArray (UIObject Type)
copy (FileSystem Type)
copy (Table Type)
copy (TCursor Type)
copyRecord
copyToArray (TCursor Type)
copyToArray (UIObject Type)
copyToToolbar
cosh
cos

D

E

empty (Array Type)
empty (Data Transfer Type)
empty (DynArray Type)
empty (Mail Type)
empty (Menu Type)
empty (Table Type)
empty (TCursor Type)
empty (Toolbar type)
empty (UIObject Type)
enableExtendedCharacters
endEdit (TCursor Type)
endEdit (UIObject Type)
end (TCursor Type)
end (TextStream Type)
end (UIObject Type)
enumAliasLoginInfo
enumAliasNames
enumAutomationServers
enumClipboardFormats
enumConstants
enumConstantValues
enumControls
enumDataBaseTables
enumDataModel
enumDesktopWindowHandles
enumDesktopWindowNames
enumDriverCapabilities
enumDriverInfo
enumDriverNames
enumDriverTopics
enumEngineInfo
enumEnvironmentStrings
enumEvents
enumExperts
enumFamily
enumFieldNamesInIndex (Table Type)
enumFieldNamesInIndex (TCursor Type)
enumFieldNames (Table Type)
enumFieldNames (TCursor Type)
enumFieldNames (UIObject Type)
enumFieldStruct (Query Type)
enumFieldStruct (Table Type)
enumFieldStruct (TCursor Type)
enumFileList
enumFolder
enumFonts
enumFormats
enumFormNames
enumForms
enumInbox
enumIndexStruct (Table Type)
enumIndexStruct (TCursor Type)
enumLocks (TCursor Type)
enumLocks (UIObject Type)
enumMethods
enumObjectNames

enumObjects
enumOpenDatabases
enumPrinters
enumProperties
enumRefIntStruct (Table Type)
enumRefIntStruct (TCursor Type)
enumRegistryKeys
enumRegistryValueNames
enumReportNames
enumRTLClassNames
enumRTLConstants
enumRTLErrors
enumRTLMethods
enumSecStruct (Table Type)
enumSecStruct (TCursor Type)
enumServerClassNames
enumServerInfo
enumSource (Form Type)
enumSource (Library Type)
enumSource (UIObject Type)
enumSourcePageList
enumSourceRangeList
enumSourceToFile (Form Type)
enumSourceToFile (Library Type)
enumSourceToFile (UIObject Type)
enumTableLinks
enumTableProperties
enumUIClasses
enumUIObjectNames (Form Type)
enumUIObjectNames (Report Type)
enumUIObjectNames (UIObject Type)
enumUIObjectProperties (Form Type)
enumUIObjectProperties (Report Type)
enumUIObjectProperties (UIObject Type)
enumUsers
enumVerbs
enumWindowHandles
enumWindowNames
eof
eot
errorClear
errorCode (Event Type)
errorCode (System Type)
errorHasErrorCode
errorHasNativeErrorCode
errorLog
errorMessage
errorNativeCode
errorPop
errorShow
errorTrapOnWarnings
exchange
execMethod (Library Type)
execMethod (UIObject Type)
execute (DDE Type)
execute (System Type)

F

G

H

I

id (MenuEvent Type)
ignoreCaseInLocate
ignoreCaseInStringCompares
importASCIIFix
importASCIIVar
importSpreadsheet
indexOf
index
initRecord
insert
insertAfter
insertAfterRecord (TCursor Type)
insertAfterRecord (UIObject Type)
insertBefore
insertBeforeRecord (TCursor Type)
insertBeforeRecord (UIObject Type)
insertFirst
insertObject
insertRecord (TCursor Type)
insertRecord (UIObject Type)
insertRow
insertTable
instantiateView
int
invoke
isAbove
isAdvancedWildcardsInLocate
isAltKeyDown
isAppBarVisible
isAssigned (AddinForm Type)
isAssigned (AnyType Type)
isAssigned (Database Type)
isAssigned (Query Type)
isAssigned (SQL Type)
isAssigned (Session Type)
isAssigned (Table Type)
isAssigned (TCursor Type)
isBelow
isBlank
isBlankZero
isCompileWithDebug (Form Type)
isContainerValid
isControlKeyDown (KeyEvent Type)
isControlKeyDown (MouseEvent Type)
isCreateAuxTables
isDesign
isDir
isEdit (TCursor Type)
isEdit (UIObject Type)
isEmpty (Query Type)
isEmpty (String type)
isEmpty (Table Type)
isEmpty (TCursor Type)
isEmpty (UIObject Type)
isEncrypted (Table Type)
isEncrypted (TCursor Type)

J-K

L

M

N

O

P

Q

qLocate
query

R

rand
readChars
readEnvironmentString
readFromClipboard (Binary Type)
readFromClipboard (Graphic Type)
readFromClipboard (Memo Type)
readFromClipboard (OLE Type)
readFromClipboard (String Type)
readFromFile (Binary Type)
readFromFile (Graphic Type)
readFromFile (Memo Type)
readFromFile (Query Type)
readFromFile (SQL Type)
readFromRTFFile (Memo Type)
readFromString (Query Type)
readFromString (SQL Type)
readLine
readMessage
readProfileString
reason (ErrorEvent Type)
reason (Event Type)
reason (MenuEvent Type)
reason (MoveEvent Type)
reason (StatusEvent Type)
recNo
recordStatus (TCursor Type)
recordStatus (UIObject Type)
registerControl
reIndexAll (Table Type)
reIndexAll (TCursor Type)
reIndex (Table Type)
reIndex (TCursor Type)
remove (Array Type)
remove (Menu Type)
remove (Toolbar Type)
removeAlias
removeAllItems
removeAllPasswords
removeButton
removeCriteria
removeItem (Array Type)
removeItem (DynArray Type)
removeMenu
removePassword
removeProjectAlias
removeRow
removeTable
rename (FileSystem Type)
rename (Table Type)
replaceItem
resourceInfo

S

setFieldValue
setFileAccessRights
setFlyAwayControl
setGenFilter (Table Type)
setGenFilter (TCursor Type)
setGenFilter (UIObject Type)
setIcon
setId (ActionEvent Type)
setId (MenuEvent Type)
setIndex (Table Type)
setInside
setItem
setKeyviol
setLeftDown
setMaxRows
setMenuChoiceAttributeById
setMenu (Form Type)
setMenu (Report Type)
setMessage
setMessageType
setMiddleDown
setMousePosition
setMouseScreenPosition
setMouseShapeFromFile
setMouseShape
setNewValue
setPosition (AddinForm Type)
setPosition (Form Type)
setPosition (Toolbar Type)
setPosition (TextStream Type)
setPosition (UIObject Type)
setPrivDir
setProblems
setProperty (AddinForm Type)
setProperty (UIObject Type)
setProtoProperty
setQueryRestartOptions
setRange (Table Type)
setRange (TCursor Type)
setRange (UIObject Type)
setReadOnly
setReason (ErrorEvent Type)
setReason (Event Type)
setReason (MenuEvent Type)
setReason (MoveEvent Type)
setReason (StatusEvent Type)
setRegistryValue
setRetryPeriod
setRightDown
setRowOp
setSelectedObjects
setShiftKeyDown (KeyEvent Type)
setShiftKeyDown (MouseEvent Type)
setSize
setSource
setSourceCharSet

setSourceDelimitedFields
setSourceFieldNamesFromFirst
setSourceRange (Data Transfer Type)
setSourceSeparator
setState
setStatusValue
setStyleSheet
setSubject
setTimer
setTitle (AddinForm Type)
setTitle (Form Type)
setUserLevel
setVCharCode
setVChar
setWorkingDir
setX (MouseEvent Type)
setX (Point Type)
setXY
setY (MouseEvent Type)
setY (Point Type)
shortName
showDeleted (Table Type)
showDeleted (TCursor Type)
show (AddinForm Type)
show (Form Type)
show (Menu Type)
show (PopUpMenu Type)
show (Toolbar Type)
showApplicationBar
showToolbar
sinh
sin
size (Array Type)
size (Binary Type)
size (DynArray Type)
size (FileSystem Type)
size (String Type)
size (TextStream Type)
sizeEx (String Type)
skip (TCursor Type)
skip (UIObject Type)
sleep
smallInt
sort
sortTo
sound
space
splitFullFileName
sqrt
startUpDir
statusValue
string
strVal
substr
subtract (Table Type)
subtract (TCursor Type)

T

U

V

view (Array Type)
view (AnyType Type)
view (DynArray Type)
view (Record Type)
view (UIObject Type)
vkCodeToKeyName

W

wait (AddinForm Type)
wait (Form Type)
wait (TableView Type)
wantInMemoryTCursor (Query Type)
wantInMemoryTCursor (SQL Type)
wasLastClicked
wasLastRightClicked
windowClientHandle
windowHandle (AddinForm Type)
windowHandle (Form Type)
windowsDir
windowsSystemDir
winGetMessageID
winPostMessage
winSendMessage
workingDir
writeEnvironmentString
writeLine
writeProfileString
writeQBE
writeSQL
writeString
writeToClipboard (Binary Type)
writeToClipboard (Graphic Type)
writeToClipboard (Memo Type)
writeToClipboard (OLE Type)
writeToClipboard (String Type)
writeToFile (Binary Type)
writeToFile (Graphic Type)
writeToFile (Memo Type)
writeToRTFFile (Memo Type)

X

x (MouseEvent Type)
x (Point Type)

Y-Z

y (MouseEvent Type)
y (Point Type)
year

{button ,AL(`LISTS;',0,"Defaultoverview",)} **Related Topics**

ObjectPAL glossary

A

active
alias
alpha operator
ANSI
application
argument
array
array element
ASCII

B

BDE
blank
braces
branching commands
breakpoint
bubbling
built-in event method

C

cast
column
comparison operator
compound object
constant
container
containership

S

T

U-Z

Active

A built-in object variable that represents the currently active object—the last object to receive focus from a **moveTo** method. Typically, the active object is highlighted. Even when focus is removed from an object (e.g., to activate another form), Active still refers to that object. Active is only resent when you move off that object.

General routines can be written to operate on an active object without specifying the particular object. For example, if a form contains two table frames, each bound to a different table, the following statement automatically operates on the active table frame:

```
active.action(DataNextRecord)
```

alias

A name you assign to a full path name to simplify access. For example, if you assign the alias MYDIR to C:\DATA\DEMOAPP\, then you refer to MYDIR to access the folder, instead of typing the whole path.

alpha operator (+)

Used to concatenate two alpha or memo fields.

ANSI

(American National Standards Institute) A sequence of 8-bit codes that defines 256 standard characters, letters, numbers, and symbols. The ASCII character set includes the first 128 ANSI characters.

application

An ObjectPAL type you use to get a handle for a Corel Paradox application. A handle is a unique variable identifier.

OR

A group of forms, methods, queries, and procedures that form a single unit, where users can enter, view, maintain, and report their data.

argument
A variable, constant, or expression that you pass to a method or procedure (also called a formal parameter).

array

An ordered set of data elements of the same data type. Array elements (also called items) are specified by a subscript that is enclosed in square brackets. For example, such that ar[1] and ar[2] are the first two elements of an array named *ar*. Subscripts are also called indexes.

Array is an ObjectPAL data type.

Note

- Array subscripts begin with 1 in ObjectPAL.

array element

One item in an array, specified by the array name and a subscript enclosed in square brackets. For example, the array Ar created with the following declaration has seven string elements, ar[1] through ar[7]:

```
ar Array [7] String
```

The following reference refers to the third element in Ar:

```
Ar[3]
```

Note

- Array subscripts begin with 1 in ObjectPAL.

ASCII

(American Standard Code for Information Interchange) A sequence of 7-bit codes that define 128 standard characters, letters, numbers, and symbols. ASCII codes have been extended to 8-bit ANSI codes that include special graphic characters. ANSI codes are used by Windows products.

BDE

(Borland Database Engine) A database engine that Corel Paradox and other Corel Windows products are based on. BDE allows you to share external database tables and other files directly with other Corel Windows products. Information about your PC's environment is maintained in the BDE configuration file (IDAPI.CFG).

blank
A field or variable that has no value.

braces

The symbols { and }. Braces enclose comments in ObjectPAL code.

branching commands

Commands that determine which ObjectPAL statements are executed or whether they are executed at all, depending on whether the conditions that they specify are met. if, iif, and switch are branching commands.

breakpoint

A flag you set in source code that is used in debugging to suspend execution.

When code that is executing reaches a breakpoint, execution is suspended. Breakpoints allow you to inspect the values of selected variables and trace the code statements that have already executed.

bubbling

A process by which events pass from the target object up through the containership hierarchy.

When bubbling occurs, an external event, directed at a target object that does not have a method to process it, passes up through the containership hierarchy until it reaches an object that can process it, or the form level. If the event can't be processed at the form level, it dies.

built-in event method

Predefined code attached to every object you place in a form. Built-in event methods define an object's default response to events.

cast

To convert a value to a specified ObjectPAL data type. For example, you can cast a value as either a Number or a SmallInt type in ObjectPAL code.

column

A vertical component that contains one field in Corel Paradox tables. In Corel Paradox reports, columns are vertical areas containing one or more fields.

comparison operator

Typographic symbols that compare the values of two fields of the same data type, and returns a True or False (logical) value. The six comparison operators include: = (equal to), <> (not equal to), < (less than), > (greater than), <= (less than or equal to), and >= (greater than or equal to). = (equal to) is a comparison operator in expressions; otherwise it is an assignment operator.

compound object

An object made up of two or more other objects. For example, a table frame is a compound object composed of field objects and record objects.

constant

A value that cannot be changed. Corel Paradox also contains many predefined constants. For example, DataNextRecord is an ObjectPAL constant that specifies a move to the next record in a table. You can also create constants that are used much like variables in ObjectPAL code in a **const...endConst** block.

container

An object that completely surrounds other objects on a form. All objects on a form coexist in a hierarchy of containers. For an object to be a container, its Contain Objects property must be checked in its Design menu. A container can itself be contained by another object.

OR

A built-in object variable that represents the object that contains *Self*. The following code specifies the **pushButton** method for a button contained within a box:

```
container.color = Red
```

Because the box contains the button, and the button is executing the code, the box turns red when the code executes.

containership

The term for an object that resides completely within the borders of another object. Containership affects the availability of variables, methods, and procedures.

control structure

One of three structures that control the execution of ObjectPAL code:

- branching control structure (e.g., **if...then...endIf**)
- looping structure (e.g., while ...endWhile)
- terminating structure (e.g., quitLoop)

ObjectPAL uses the following methods as control structures: **for, return, forEach, scan, if, switch, iif, try, loop, while,** and **quitLoop**.

CTRL + Break

A key sequence that halts program execution. You can configure Corel Paradox to respond to CTRL + Break by clicking Tools, Settings, Developer Preferences and enabling the Enable CTRL + Break check box.

data
The information that Corel Paradox stores in a table.

data type

The type of data that a field, variable, or array element contains. Data types are also called *classes* in other languages.

Database

Related data and objects that are organized logically into Corel Paradox tables.

OR

An ObjectPAL variable that contains information about relationships between tables or access to the tables.

Date, Time and DateTime operator

Used to add (+) or subtract (-) values from Date, Time, and DateTime fields.

DDE

(Dynamic Data Exchange) A method of sharing data between Windows applications.

deadlock

A situation created in a multi-user environment when two incompatible lock commands are issued simultaneously.

Debugger

A component of the ObjectPAL Integrated Development Environment (IDE). The Debugger allows you to interactively find and correct errors in code by testing and tracing the execution of commands.

You can use the Debugger to inspect the values of variables at the breakpoints in your code and to trace the execution of code.

desktop
The main window in Corel Paradox.

dialog box

A form that is displayed when additional information is needed to complete an action or command. A dialog box is displayed on top of other windows and can be moved on top of the Menu Bar. A form's Form Window Properties specify that the form is a dialog box.

Display manager

A category of object types that includes Application, Form, Report, and Table View.

DLL

(Dynamic Link Library) A library of external routines that perform common tasks that Windows programs can share. DLL routines are loaded and linked to ObjectPAL methods and procedures at run time. DLLs handle user input, manage memory, and allow you to create custom routines which perform tasks that exceed ObjectPAL's functionality.

dynamic array

An array in which each item has a string for an index. For example, ["Product"], ["Corel Paradox"], ["Type"] ["Relational database"], ["Version"] [1.0]. These arrays are dynamic because their size changes as items are added and removed. The dynamic array's size is limited only by system memory.

The DynArray data type in ObjectPAL allows you to retrieve values in a large dynamic array quickly and easily.

Editor

A component of the ObjectPAL Integrated Development Environment (IDE), used to create and edit ObjectPAL methods.

encrypt

To translate a table or script into password-protected code.

error stack

An ObjectPAL mechanism that stores information about the most recently detected run-time error. When a method or procedure executes successfully, the error stack is cleared.

If a run-time error occurs, error information records are pushed onto the stack in a last-in-first-out arrangement. Error information records contain error code and error messages. The following methods allow you to control and access records on the error stack: **errorCode**, **errorMessage**, **errorPop**, **errorClear**, **errorHasErrorCode**, **errorHasNativeErrorCode**, **errorLog**, and **errorShow**.

event

An action or condition that triggers the execution of a method. Internal events are triggered by Corel Paradox. External events are triggered by the user or by an ObjectPAL method that simulates a user action.

OR

An ObjectPAL type that contains information about an event.

event model

The rules that specify how events are processed by objects in a form.

event packet

An ObjectPAL structure that contains detailed information about events (e.g., target objects and the reasons the event occurred). The event packet is passed into built-in event methods through the variable *eventInfo*, and accompanies the event as it moves up the containership hierarchy. You can use Run-Time Library (RTL) methods to examine the contents of the event packet.

event-driven application

An application whose code executes in response to events. In procedural applications, code executes in a linear sequence.

example element

An arbitrary sequence of characters in a sequence that represents any value in a field. In Corel Paradox you create an example element by clicking Example and typing the characters in the query image. In methods, you create example elements by inserting an underscore before the characters.

expression

A group of characters (e.g., data values, variables, arrays, operators, and functions) that evaluate to a single a quantity or value. An expression can evaluate to a specific data type or can be converted to string values before it is evaluated.

field

An item in a table that contains a specific category of information. Fields are displayed in tables as vertical columns. A horizontal row of fields in a table is a record.

field assignment

Refers to the assignment that links a variable to a field. When the variable changes, the value in the field changes.

field object

A UIObject. When bound to a field in a table, a field object is used to display or change field data. You can use an unbound field object to get input from the user.

field type

The representation of data in a table that is specific to the table's driver. For example, alphanumeric is a Corel Paradox field type and character is the corresponding dBASE field type.

field value

The data contained in one field of a record. If no data is present, the field is blank. Field objects have a Value property.

Field View

A display option that allows you to move the cursor through a field, character by character. Use Field View to view values that are too large to be displayed in the current field width, or to edit a field value.

file

A collection of information stored under one name on a disk. Corel Paradox tables are stored in files.

FileSystem type

An ObjectPAL variable that contains information about disk files, drives, directories and folders. A FileSystem variable provides a handle to a file or directory that you can work with in ObjectPAL statements.

focus

An object that has focus (the active object) can accept keyboard or mouse input and is usually highlighted. Only one object can have focus at a time.

form
A window that displays data and objects.
OR
An ObjectPAL data type. The form is the highest-level container object.

format specification

The way in which a field value is displayed on screen or output to a printer. Format specifications include alignment (left, right, or center), font style and size, upper or lower-case, and date and time formats. Format specifications also control the way in which input data is read, or validated.

function keys

The 12 keys across the top of the keyboard, labeled F1 through F12. Some keyboards have 10 keys at the far left of the keyboard, labeled F1 through F10.

global variable
A variable that is available to all objects in a form.

handle

A variable that uniquely identifies an object you want to manipulate in ObjectPAL.

Help

The Corel Paradox online Help system. You can press F1 at any point in Corel Paradox to display information about the current operation.

hierarchy
The relationship of objects in a form, derived from their visual, spatial relationship.

IBM extended codes

Keys or combinations of keys on the keyboard that are given extended code numbers between -1 and -132. These keyboard keys do not correspond to any of the standard ASCII character codes.

identifier

A label that refers to an object, variable, property, or value. For example, the value returned by a function is an identifier.

incremental development

An application development process in which small parts of the application, or its overall structure, are designed and tested interactively.

index

A file that determines the order in which Corel Paradox accesses the records in a table. The key field of a Corel Paradox table establishes its primary index. In some languages, an array subscript is an index.

insertion point

The place where text is inserted when you type. The insertion point is usually indicated by a flashing vertical bar, the cursor.

inspect

To right-click an object and view the corresponding pop-up menu.

key

A field or group of fields that uniquely identify each record in a Corel Paradox table. A key prevents the table from containing duplicate records, maintains the records' sorted order, and creates a primary index. A composite primary key is contains more than one field.

key field

The index key for organizing and sorting a table.

keycode

A code that represents a keyboard character in ObjectPAL methods. The keycode may be an ANSI number or a string that represents a key name in Corel Paradox.

keyword

A word reserved by ObjectPAL. A keyword cannot be used as the name of a variable, array, method, or procedure.

lastMouseClicked

A built-in object variable that represents the last object to receive a **mouseDown**. **lastMouseClicked** is reset when the mouse button is released, but only after the object has completed its **mouseUp**.

lastMouseRightClicked

A built-in object variable that represents the last object to receive a **mouseRightDown**.
lastMouseRightClicked is reset when the mouse button is released, but only after the object has completed its **mouseRightUp**.

library

ObjectPAL code that can be used by objects in one or more forms.

OR

An ObjectPAL data type that stores custom methods and procedures, variables, constants, and user-defined data types. Libraries store and maintain frequently used routines, and allow you to share custom methods and variables among several forms.

lifetime
The time that a local variable, proc, or method is active or available.

link key

The part of the subordinate table's key that is linked or matched to fields in the master table, in a linked multi-table form.

local variable

A variable that is available only to the method or procedure in which it is declared.

logical operator

One of three operators (AND, OR, or NOT) that can be used on logical data. For example, an AND between two logical values results in a logical value of True if the original values are also True. Logical operators are also known as Boolean operators.

logical type

An ObjectPAL data type that can contain one of two values: True or False. The logical type is also known as Boolean type.

logical value

A True or False value that is assigned to an expression when it is evaluated. Logical values are also known as Boolean values.

looping commands

Commands that repeat a series of actions while or until a condition is met. (e.g., for, loop, while).

menu

A display of the commands or options available. You can use ObjectPAL to create and edit application menus and pop-up menus.

The Menu Bar is a special menu at the top of the desktop. Click an item on the Menu Bar to view a list of available commands.

OR

An ObjectPAL type that stores information about an application's Menu Bar.

menu choice
A command or option selected from the Corel Paradox menus.

message
A string expression displayed in the Status Bar.

method

ObjectPAL code that is attached to an object to define the object's response to an event.

There are three types of methods:

- built-in event methods, which are attached to UIObjects and respond to events
- RTL (run-time library) methods which are part of the ObjectPAL language
- custom methods which you create when a built-in event method or RTL method does not provide adequate functionality

The syntax for a method is object.method(arguments).

modal

A dialog box that retains focus until you close it. A modal dialog box cannot be resized.

normalized database structure

A logical arrangement of database information in small tables. Normalized databases structures minimize redundancy, allow efficient access to data, provide logical and coherent views of data categories, promote data integrity, and allow easy insertion and deletion. Normalization involves decomposing a single large table into smaller tables that are linked by key fields.

numeric operator

A numeric operator that performs arithmetic operations on the surrounding operands. There are four numeric operators: + (addition), - (subtraction), * (multiplication), and / (division). Numeric operators are valid with date, time and number fields only.

object

Items such as forms, reports, tables, queries, scripts, SQL files, and libraries.

Form and report objects can contain UI objects such as boxes, lines, ellipses, text, graphics, OLE objects, buttons, fields, table frames, and multi-record objects.

ObjectPAL recognizes the following object type categories:

Data model objects	Display managers
Data types	Events
Design objects	System data objects

Object Tree

A diagram that illustrates the containership relations that exist between objets in a form.

OEM

(Original Equipment Manufacturer) Your computer's manufacturer.

OLE

(Object Linking and Embedding) A method by which an object, such as a graphic image, a spreadsheet, or a word processing document, can be linked or embedded in a Corel Paradox table or form.

OR

An ObjectPAL data type that is used with OLE objects.

parameter

The variable through which an argument is passed at run time. Parameters (also called formal parameters) are defined in a procedure declaration..

picture

A pattern of characters that defines what a user can type into a field during editing, data entry, or in response to a prompt.

pixel

A single point on the screen. The name pixel is derived from the term picture element.

point

An ordered pair of numbers that represents a location on screen.

OR

An ObjectPAL data type that contains information about a point on screen.

pointer
A visual marker that specifies the mouse position on screen.

post

To accept or commit changes to a record and update the table.

primary index

An index that is based on the key field(s) of a Corel Paradox table. A primary index determines the location of records; allows you to use the table as the detail in a link; maintains the records in sorted order; and accelerates table operations.

procedure

Code that is bracketed by the keywords **proc** and **endProc**. Unlike a method, procedures are global—not bound to an object that gives it context.

There are two types of procedures:

- RTL (run-time library) procedures which are part of the ObjectPAL language
- custom procedures, which you create when an RTL procedure does not provide adequate functionality

A procedure's syntax is **procedure(arguments)**.

prompt

Instructions displayed on the screen, usually in the Status Bar. Prompts request information or guide you through an operation.

property

A named attribute that determines one aspect of an object's behavior or characteristics. You can edit an object's properties with the Object Explorer.

QBE

(Query By Example) A method of creating a query by selecting its fields and selection criteria visually. QBE does not require formal SQL statements.

query

An inquiry about the data in a table; or an instruction to update the data, through the INSERT, DELETE or CHANGETO operators.

OR

An ObjectPAL variable that represents a QBE query.

query by example (QBE)

A method of creating a query by selecting its fields and selection criteria visually. QBE does not require formal SQL statements.

quoted string
Text that is enclosed in double quotation marks.

raster operation
An operation that specifies how colors are blended on the screen.

record

A horizontal row in a Corel Paradox table that contains a group of related fields of data.

OR

An ObjectPAL data type: a programmatic, user-defined collection of information, similar to a **record** in Pascal or a **struct** in C. Separate and distinct from records associated with a table.

record number

A unique number that identifies each record in a table.

relational database

A database modeled after a set of principles called the relational model. Data in a relational database must be organized into tables.

reserved words

The names of commands, keywords, functions, system variables, and operators in Corel Paradox or ObjectPAL language. Reserved words may not be used as ObjectPAL variables or array names.

restricted view

A detail table on a multi-table form. A restricted view is linked to the master table on a one-to-one or one-to-many basis and shows only those records that match the current master record.

row

A horizontal component of a Corel Paradox table that contains a record.

run-time error
An error that occurs when a syntactically valid statement cannot be executed.

run-time library (RTL)
Predefined methods and procedures that operate on specific objects.

scope

The availability of a variable, method, or procedure to other objects, methods. or procedures.

Variables, methods, and procedures are attached to objects. When an object becomes inactive (e.g., execution is complete) the attached variables, methods, and procedures also become inactive. A variable that is declared inside a method is said to go out of scope when that method completes. The variable has no existence outside the method's operation.

script

A collection of ObjectPAL statements that you use to perform operations automatically. Scripts are usually attached to an object on a form and are also called macros.

secondary index
An index used for linking, querying, and changing the view order of tables.

Self

A built-in object variable that represents the UIObject to which the currently executing code is attached. For example, when the following statement executes in the **mouseEnter** method attached to *theBox*, *Self* specifies the UIObject called *theBox*.

```
self.color = Red
```

Suppose that a method that is attached to *theBox* calls a custom method named **changeColor,** that is itself attached to the page. The code for **changeColor** is

```
method changeColor()

self.color = Blue

endMethod
```

When the method attached to *theBox* calls **changeColor**, the page turns blue, not *theBox*. In this case, *Self* refers to the object to which the code is attached regardless of which object called the code. However, when *Self* appears in a statement in a library, *Self* refers to the object that called the library routine.

When an event occurs, *Self* and **eventInfo.getTarget** can refer to the same object; however, as events bubble up the containership chain, the target remains fixed while *Self* changes to refer to the object that executes the method.

Self does not refer to the object's value or name.

session

A channel to the database engine. A session occurs whenever you open Corel Paradox, either interactively or using ObjectPAL. You can have multiple sessions running simultaneously.

OR

An ObjectPAL object type which opens additional sessions.

slash sequence

A backslash followed by one or more characters that is used to represent an ASCII character. Examples are \" or \ 018. You can use slash sequences to place quotation marks within strings and to include other characters that have special meaning to Corel Paradox.

standalone script

A script that is not attached to an object in a form and is run directly from the desktop.

Status Bar

A row of four windows displayed across the bottom of the desktop. Use the **reason** or **setReason** methods and the StatusReasons constants to determine where a message is displayed.

string

An alphanumeric value or an expression consisting of alphanumeric characters.

OR

An ObjectPAL data type.

structure
The arrangement of fields in a table.

subject

A built-in object variable that specifies which object a custom method should operate on. Suppose that a page in a form has a custom method **setColor**. The code for **setColor** is:

```
method setColor()

   subject.color = red

endMethod
```

If an object on the page makes the following call, the object named *someObject* will turn red:

```
someObject.setColor()
```

When **setColor** executes, it replaces *Subject* with *someObject*.

substring
Any part of a string.

syntax error

An error that occurs due to an incorrectly expressed statement.

table

A structure composed of horizontal rows (records) and vertical columns (fields). Tables contain stored information.

table alias
Another name for a table in a data model.

TableView

An ObjectPAL data type. Use TableView to get a handle to a table view, the representation of a table in rows and columns.

target
The object for which an event is intended. For example, when you click a button, the button is the target.

TCursor

An ObjectPAL type that points to the data in a table. Using TCursors, you can manipulate the data without displaying the corresponding table.

tilde variable

A variable that is preceded by a tilde (~). Tilde variables are used in queries.

Toolbar

A collection of buttons and design tools displayed below the Menu Bar. The buttons available in the Toolbar correspond to the active object.

transaction
A group of related changes to a database.

trapping

ObjectPAL watches for a specified event as it executes code. If the event occurs, ObjectPAL handles it with a custom code routine. Trapping is especially useful for errors. You can specify your own code to deal with the error, so it won't interfere with the execution of the rest of your code.

twip

A unit of measurement equal to 1/1440 of a logical inch (or 1/20 of a printer's point). There are 567 twips in one centimeter.

type

A method of classifying objects with similar attributes. For example, all tables have common attributes, as do all forms, but the attributes of tables and forms are different, tables and forms belong to different types.

validity check

A constraint on the values that you can enter in a field. Validity checks are also called val checks.

variable

A placeholder in memory where data is temporarily stored and manipulated while ObjectPAL code is running. Variables must have names that are unique within their scope.

Variables that are declared by a **var** statement run faster (and often use less space) than undeclared variables. Declaring variables also enables compiler-type verification, which helps detect bugs before run time.

OleAuto type

OLE Automation allows you to manipulate an application's objects from outside that its application. OLE Automation uses OLE's component object model, but can be implemented independently from the rest of OLE. You can use OleAuto methods to create and manipulate objects from an application that exposes objects to OLE.

Methods for the OleAuto type

attach
close
enumAutomationServers
enumConstants
enumConstantValues
enumControls
enumEvents
enumMethods
enumObjects
enumProperties
enumServerInfo
first
invoke
next
open
openObjectTypeInfo
openTypeInfo
registerControl
unregisterControl
version

{button ,AL(`OPAL_TYPE_OLEAUTO;OPAL_Pdox_OLEAuto_server;',0,"Defaultoverview",)} **Related Topics**

Print related ObjectPAL methods and examples

Using Corel Paradox 8 as an OLE Automation server

When working in another application, you can use the following OleAuto commands with Corel Paradox as the OLE Automation server. Most of these commands also have an equivalent ObjectPAL method.

Commands for using Corel Paradox as an OLE Automation server

Copy(filename String) LongInt
DataModelDesign(filename String) LongInt
Delete(filename String) LongInt
executeString (scriptStr String, usesStr String) String
FormDesign(filename String) String
FormNew() LongInt
FormOpen(filename String) LongInt
GetLastError() LongInt
GetLAstErrorAsString()String
GetLastErrorAtDepth(depth LongInt) LongInt
GetLastErrorStackedDepth() LongInt
LibraryDesign(filename String) LongInt
LibraryNew() LongInt
LibraryOpen(filename String) LongInt
MsgBox(strTitle String, strMsg String) LongInt
PrinterSheetDesign(filename String) LongInt
QueryDesign(filename String) LongInt
QueryNew() LongInt
QueryOpen(filename String) LongInt
Quit()
Rename(filename String) LongInt
ReportDesign(filename String) LongInt
ReportNew() LongInt
ReportOpen(filename String) LongInt
ReportPrint(filename String) LongInt
ScreenSheetDesign(filename String) LongInt
ScriptDesign(filename String) LongInt
ScriptNew() LongInt
ScriptOpen(filename String) LongInt
ShowLastErrorStack() LongInt
SQLQueryDesign(filename String) LongInt
SQLQueryNew() LongInt
SQLQueryOpen(filename String) LongInt
TableAdd(filename String) LongInt
TableEmpty(filename String) LongInt
TableExport(filename String) LongInt
TableInfoStructure(filename String) LongInt
TableOpen(filename String) LongInt
TableRestructure(filename String) LongInt
TableSort(filename String) LongInt
TableSubtract(filename String) LongInt

attach method

Attaches an OLE Automation variable to a UIObject.

Syntax

```
attach ( const object UIObject ) Logical
```

Description

attach attaches an OLE Automation variable to the UIObject specified by *object*. **attach** succeeds if the UIObject denotes an ActiveX control. When **attach** succeeds, the objects methods and properties are accessible from the OLE Automation variable.

Example

{button ,AL(`OPAL_TYPE_OLEAUTO;OPAL_METH_OACLOSE;OPAL_METH_OAOPEN;OPAL_METH_OAREGISTERCONTROL;OPAL_METH_OAVERSION;',0,"Defaultoverview",)} Related Topics

attach example

The following example attaches to an OLE custom control called MyCtrl that is embedded on the form:

```
method pushButton ( var eventInfo Event )
var
   oa   oleauto
endvar
   oa.attach(MyCtrl)
endMethod
```

close method

Closes the OLE Automation variable.

Syntax

close ( ) Logical

Description

close releases the reference from an OLE Automation variable to an automation server; however, some servers remain open when all references are removed. **close** is especially useful for global variables, because it is called automatically when an OLE Automation variable goes out of scope.

Example

{button ,AL(`OPAL_TYPE_OLEAUTO;OPAL_METH_OAOPEN;',0,"Defaultoverview",)} **Related Topics**

close example

The following example closes the OLE Automation server application:

```
var
   pdx oleauto
endvar
method pushButton ( var eventInfo Event )
   pdx.close()
endMethod
```

enumAutomationServers procedure

Reads the registry on the current machine and lists the available OLE Automation servers.

Syntax

```
enumAutomationServers ( var servers DynArray[ ] String ) Logical
```

Description

enumAutomationServers lists the OLE Automation servers and OLE custom controls in the registry.

The information is assigned to *servers*, a dynamic array that you must declare and pass as an argument. The indexes of the DynArray are the end user OLE Automation server names. The corresponding index values are the internal OLE names (e.g., Paradox.Application).

enumAutomationServers returns True if successful; otherwise, it returns False.

Use **enumAutomationservers** to retrieve the internal server name to pass to **open** and **openTypeInfo**.

Example

{button ,AL(`OPAL_TYPE_OLEAUTO;OPAL_METH_OAENUMSERVERINFO;OPAL_METH_OAOPEN;OPAL_METH_OAOPENOBJECTTYPEINFO;OPAL_METH_OAOPENTYPEINFO;',0,"Defaultoverview",)} Related Topics

enumAutomationServers example

The following example demonstrates how **enumAutomationServers** compiles a list of OLE Automation servers:

```
method pushButton ( var eventInfo Event )
var
   da DynArray[] String
endVar
enumautomationservers(da)
da.view()
endMethod
```

enumConstants method

Enumerates the constants defined by an OLE Automation server.

Syntax

```
enumConstants ( var types DynArray[ ] String ) Logical
```

Description

enumConstants enumerates the constant type names in a type library of an OLE Automation server. The information is assigned to the dynamic array (DynArray) *types*. The indexes hold the OLE type name and the corresponding items are the equivalent ObjectPAL type. You can use the constant type name as input for the **enumConstantValues** to retrieve the constant values of this type. These constants are only available through this method.

Example

{button ,AL(`OPAL_TYPE_OLEAUTO;OPAL_METH_OAENUMCONSTANTVALUES;OPAL_METH_OAENUMCONTROLS;OPAL_METH_OAENUMEVENTS;OPAL_METH_OAENUMMETHODS;OPAL_METH_OAENUMOBJECTS;OPAL_METH_OAENUMPROPERTIES;',0,"Defaultoverview",)} Related Topics

enumConstants example

The following example enumerates the constants from Excel:

```
method pushButton ( var eventInfo Event )
   var
      oa oleauto
      da DynArray[] String
   endvar
   oa.open("Excel.application.5")
   oa.enumConstants(da)
   da.view("Excel constant types")
endmethod
```

enumConstantValues method

Enumerates the constants that are accessible from an OLE Automation server.

Syntax

```
enumConstantValues ( const constantType String, var values DynArray[ ] AnyType ) Logical
```

Description

enumConstantValues enumerates the constants in a type library of an OLE Automation object. *constantType* is the type returned by **enumConstants**.

The enumerated information is assigned to the dynamic array (DynArray) *values*. The indexes are the OLE constant names and the corresponding items are the constant's values.

Example

{button ,AL(`OPAL_TYPE_OLEAUTO;OPAL_METH_OAENUMCONSTANTS;OPAL_METH_OAENUMCONSTANTVALUES;OPAL_METH_OAENUMCONTROLS;OPAL_METH_OAENUMEVENTS;OPAL_METH_OAENUMMETHODS;OPAL_METH_OAENUMOBJECTS;OPAL_METH_OAENUMPROPERTIES;',0,"Defaultoverview",)} Related Topics

enumConstantValues example

The following example enumerates the values of constants available in Excel:

```
method pushButton ( var eventInfo Event )
var
   oa oleauto
   da DynArray[] AnyType
endvar
   oa.open("Excel.Application.5")
   oa.enumConstantValues("Constants",da)
   da.view()
endmethod
```

enumControls procedure

enumControls enumerates the registered OLE custom controls.

Syntax

```
enumControls ( var controls DynArray[ ] String ) Logical
```

Description

enumControls enumerates the OLE custom controls listed in the registry. The information is assigned to the dynamic array (DynArray) *controls*. The DynArray indexes are the end user OLE Automation control names (e.g., "My Own Control"), and the corresponding values are the internal OLE names (e.g., MyCtrl.Ctrl1).

Use **enumControls** to retrieve internal ActiveX names, as input for the **open** and **openTypeInfo** methods. You can also use **enumControls** for the progid property for the OLE object.

Example

{button ,AL(`OPAL_TYPE_OLEAUTO;OPAL_METH_OAENUMCONSTANTS;OPAL_METH_OAENUMCONSTANTVALUES;OPAL_METH_OAENUMEVENTS;OPAL_METH_OAENUMMETHODS;OPAL_METH_OAENUMOBJECTS;OPAL_METH_OAENUMPROPERTIES;',0,"Defaultoverview",)} Related Topics

enumControls example

The following example builds and displays the Controls dynamic array (DynArray):

```
method pushbutton ( var eventInfo Event )
var
   da DynArray[] String
endvar
   enumControls(da)
   da.view()
endmethod
```

The following example creates a form using an ActiveX control object. MyCtrl.Ctrl1 is an internal ActiveX control name listed by **enumControls** in the previous example.

```
method pushButton ( var eventInfo Event )
var
   f form
   o uiobject
endvar
   f.create()
   o.create(OLETool, 200, 300, 1000, 500, f)
   o.ProgId = "MyCtrl.Ctrl1"
endMethod
```

enumEvents method

Enumerates the events that are accessible from an OLE Automation server.

Syntax

```
enumEvents ( var events DynArray[ ] String ) Logical
```

Description

enumEvents enumerates a controls events. The information is assigned to the dynamic array (DynArray) *events*. The DynArray is empty if the OLE Automation variable is bound to an object that is not an OLE Automation control.

Example

{button ,AL(`OPAL_TYPE_OLEAUTO;OPAL_METH_OAENUMCONSTANTS;OPAL_METH_OAENUMCONSTANTVALUES;OPAL_METH_OAENUMCONTROLS;OPAL_METH_OAENUMMETHODS;OPAL_METH_OAENUMOBJECTS;OPAL_METH_OAENUMPROPERTIES;',0,"Defaultoverview",)} Related Topics

enumEvents example

The following example opens the type library of MyCtrl.Ctrl1, and builds and displays the dynamic array (DynArray) of the enumerated events:

```
method pushButton ( var eventInfo Event )
var
   oa oleauto
   dy DynArray[] String
endvar
   oa.openTypeInfo("MyCtrl.Ctrl1")
   oa.enumEvents(dy)
   dy.view()
endMethod
```

enumMethods method

Enumerates the methods that are accessible from an OLE Automation server.

Syntax

enumMethods (var *methods* DynArray[ ] String) Logical

Description

enumMethods enumerates the methods that can be accessed from an OLE Automation server. The information is assigned to the dynamic array (DynArray) *methods*. The index of the DynArray is the method name, and its value is the ObjectPAL prototype. Some of these methods might not be accessible by ObjectPAL because their types are not supported, in which case the prototype displays an asterisk character (*).

You can specify argument types with commentary information. For example, MoveCursorToPos(x LongInt {OLE_XPOS_PIXELS}, y LongInt {OLE_YPOS_PIXELS}), where OLE_XPOS_PIXELS is the OLE type of the argument. The OLE type name often indicates the nature of the argument.

Example

{button ,AL(`OPAL_TYPE_OLEAUTO;OPAL_METH_OAENUMCONSTANTS;OPAL_METH_OAENUMCONSTANTVALUES;OPAL_METH_OAENUMEVENTS;OPAL_METH_OAENUMOBJECTS;OPAL_METH_OAENUMPROPERTIES;',0,"Defaultoverview",)} Related Topics

enumMethods example

The following example builds and displays the dynamic array (DynArray) of the enumerated methods:

```
method viewMethods(var oa oleauto)
var
   dy DynArray[] String
endvar
   oa.enumMethods(dy)
   dy.view()
endMethod
```

enumObjects method

Enumerates the events accessible from an OLE Automation server.

Syntax

```
enumObjects ( var objects DynArray[ ] String ) Logical
```

Description

enumObjects lists the names of objects in a type library of a server. The object names are sub-objects in that particular OLE server. The sub-objects are often retrieved through methods and properties of the Application server object retrieved with the **open** method. This method lists the object names, which can be passed into **openObjectTypeInfo**, from which the methods and properties of the sub-object can be enumerated.

Example

{button ,AL(`OPAL_TYPE_OLEAUTO;OPAL_METH_OAENUMSERVERINFO;OPAL_METH_OAOPENOBJECTTYPEINFO;OPAL_METH_OAOPENTYPEINFO;',0,"Defaultoverview",)} **Related Topics**

enumObjects example

The following example builds and displays the dynamic array (DynArray) of the enumerated objects.

```
method viewObjects ( oa oleauto )
var
   dy DynArray[] String
endvar
   oa.enumObjects(dy)
   dy.view()
endmethod
```

enumProperties method

Enumerates the properties accessible from an OLE Automation server.

Syntax

```
enumProperties ( var properties DynArray[ ] String ) Logical
```

Description

enumProperties enumerates the properties that can be accessed from an OLE Automation server. The information is assigned to the dynamic array (DynArray) *properties*. The index of the DynArray is the property name, and the corresponding item is the ObjectPAL type. Some properties aren't accessible by ObjectPAL because their types are not supported. Unsupported ObjectPAL types display an asterisk (*).

Property types might be specified with commentary information. For example, ForeColor LongInt {OLE_COLOR}, BackColor LongInt {OLE_COLOR}, where OLE_COLOR is the OLE type of the argument. The OLE type name often indicates the nature of the argument.

Example

{button ,AL(`OPAL_TYPE_OLEAUTO;OPAL_METH_OAENUMCONSTANTS;OPAL_METH_OAENUMCONSTANTVALUES;OPAL_METH_OAENUMEVENTS;OPAL_METH_OAENUMMETHODS;OPAL_METH_OAOPENOBJECTTYPEINFO;OPAL_METH_OAOPENTYPEINFO;',0,"Defaultoverview",)} Related Topics

enumProperties example

The following example builds and displays a dynamic array (DynArray) of the enumerated properties:

```
method viewProperties(oa oleauto)
var
   dy DynArray[] String
endvar
   oa.enumProperties(dy)
   dy.view()
endMethod
```

enumServerInfo procedure

Enumerates information about the OLE Automation server.

Syntax

```
enumServerInfo ( const serverName String, var info DynArray[ ] AnyType ) Logical
```

Description

enumServerInfo enumerates information about the server from the registry. The *serverName* is one of the internal OLE server names returned from either **enumAutomationServers** or **enumControls**.

The following table displays the information enumerated by **enumServerInfo**:

Key	Type	Comment
CLSID	String	The ClassID used internally by OLE. If CLSID exisits the server is an ActiveX control.
ProgID	String	The internal OLE server name (e.g., Paradox.Application).
TypeLib	String	The ClassID of the type library. If TypeLib exists, openTypeInfo can be used with this server.
ToolboxBitmap32	Graphic	Toolbar bitmap for the control
Version	String	The internal version of this server

Because the *info* dynamic array (DynArray) only holds information retrieved from the registry, **numServerInfo's** results depend the server's registry.

Example

{button ,AL(`OPAL_TYPE_OLEAUTO;OPAL_METH_OAENUMAUTOMATIONSERVERS;OPAL_METH_OAOPENOBJECTTYPEINFO;OPAL_METH_OAOPENTYPEINFO;',0,"Defaultoverview",)} Related Topics

enumServerInfo example

The following example builds and displays a dynamic array (DynArray) of the server information:

```
method pushButton ( var eventInfo Event )
var
   da DynArray[] anytype
endvar
   enumServerInfo("MyCtrl.Ctrl1", da)
   da.view()
endMethod
```

first method

Returns the first object in a collection.

Syntax

```
first ( var AnyType )
```

Description

first returns the first object in a collection when an OLE Automation variable denotes a sub-object in a server that is itself a collection of other sub-objects. The items in a collection are primarily OleAuto type a reference to another OLE automation object. If the collection is empty, **first** returns a blank value. You can determine is an object is a collection object using **isBlank.** If the object is a collection, **isBlank** succeeds; otherwise, it fails. Some servers do not support **isBlank.**

A collection object behaves like any other OleAuto object. It always has a Count property and an Item method, and most of the time they have an Add and a Remove method. Specific implementations can have other methods and properties available.

Example

{button ,AL(`OPAL_TYPE_OLEAUTO;OPAL_METH_OANEXT;OPAL_METH_OAOPEN;OPAL_METH_OAVERSION;',0,"Defaultoverview",)} **Related Topics**

first example

The following example uses **first** and **next** to return individual elements of an OleAuto collection object *collectnObj*. *collectnObj* is passed as a parameter to **first**. This example assumes that *collectnObj* is attached to an OleAuto collection object and is declared at the form level.

```
method GetContents ( var collectnObj oleAuto )
Var
   anElement  LongInt
   i          SmallInt
endVar

collectnObj = OA.first(anElement)
processObject(anElement)            ; call a custom method which
                                    ; processes the current element
                                    ; and pass the element reference
for i = 1 to collectnObj^count()    ; call the OleAuto object's count
                                    ; method to determine loop max
  collectnObj = OA.next(anElement)  ; get the next item in the
                                    ; collection object
  processObject(anElement)

endFor
endMethod
```

invoke procedure

Invokes a method or property in an OLE Automation server.

Syntax

```
invoke ( const methodName String [, var arg]* ) AnyType
```

Description

invoke allows you to access methods and properties in an OLE Automation server. The *methodName* argument specifies the OLE Automation server's internal method and the optional *arg* arguments are the parameters of the method specified by *methodName*.

invoke is especially useful when the OLE Automation server has a method or property name that conflicts with an ObjectPAL keyword.

Example

{button ,AL(`OPAL_TYPE_OLEAUTO;OPAL_METH_OAATTACH;OPAL_METH_OAOPEN;',0,"Defaultoverview",)} **Related Topics**

invoke example

The following example demonstrates three ways to call the **msgbox** method of the passed automation server:

```
method callMsgBox (oa oleauto)
var
   ret LongInt
endvar
   ret = oa.msgbox("Hello", 5)
   ret = oa^msgbox("Hello", 5)


   ret = oa.invoke("msgbox", "Hello", 5)
endMethod
```

next method

Returns the next object in a collection.

Syntax

```
next ( var AnyType )
```

Description

next returns the next object in a collection when an OLE Automation variable denotes a sub-object in a server that is itself a collection of other sub-objects. When there are no more items in the collection, the result will be a blank value. The items in a collection are primarily OleAuto type a reference to another OLE automation object. If the collection is empty, **next** returns a blank value. You can determine is an object is a collection object using **isBlank.** If the object is a collection, **isBlank** succeeds; otherwise, it fails. Some servers do not support **isBlank.**

A collection object behaves like any other OleAuto object. It always has a Count property and an Item method, and most of the time they have an Add and a Remove method. Specific implementations can have other methods and properties available.

Example

{button ,AL(`OPAL_TYPE_OLEAUTO;OPAL_METH_OAFIRST;OPAL_METH_OAOPEN;OPAL_METH_OAVERSION;',0,"Defaultoverview",)} Related Topics

next example

See the **first** example.

open method

Opens a server.

Syntax

```
open ( const serverName String ) Logical
```

Description

open opens the server specified by *serverName*. If the specified server is an automation server, open succeeds; otherwise, it fails.

Example

{button ,AL(`OPAL_TYPE_OLEAUTO;OPAL_METH_OAATTACH;OPAL_METH_OACLOSE;OPAL_METH_OAENUMAUTOMATIONSERVERS;OPAL_METH_OAVERSION;',0,"Defaultoverview",)} Related Topics

open example

The following example opens Corel Paradox as an OLE Automation server;

```
var
   pdx oleauto
endvar
method pushbutton ( var eventInfo Event )
   pdx.open("Paradox.Application")
endMethod
```

openObjectTypeInfo method

Enumerates the events that are accessible from an OLE Automation server.

Syntax

```
openObjectTypeInfo ( const server OleAuto, const objectName String ) Logical
```

Description

openObjectTypeInfo connects to the type library of the specified sub-object in a server. Unlike **openTypeInfo**, **openObjectTypeInfo** allows you to use **enumMethods** and **enumProperties** to retrieve the methods and properties of the sub-object specified in *objectName*. The object names can be enumerated by **enumObjects**.

Example

{button ,AL(`OPAL_TYPE_OLEAUTO;OPAL_METH_OAENUMMETHODS;OPAL_METH_OAENUMOBJECTS;OPAL_METH_OAENUMPROPERTIES;OPAL_METH_OAENUMSERVERINFO;OPAL_METH_OAOPENTYPEINFO;',0,"Defaultoverview",)} Related Topics

openObjectTypeInfo example

The following example connects to the type library of the sub-object, chart, in Excel. The code then builds and displays a dynamic array (DynArray) of the chart's properties.

```
method pushButton ( var eventInfo Event )
var
   oa oleauto
   excel oleauto
   chart oleauto
   da DynArray[] String
endvar
   excel.openTypeInfo("Excel.application.5")
   chart.openObjectTypeInfo(excel, "chart")
   chart.enumProperties(da)
   da.view()
endMethod
```

openTypeInfo method

Opens the type library of an OLE Automation server.

Syntax

```
openTypeInfo ( var serverName String ) Logical
```

Description

openTypeInfo connects to the type library of the server specified by *serverName*. Once connected, you can call the type enumeration methods to retrieve information about the server. The **openTypeInfo** method creates an instance of the server and gives you access to the server methods and properties. If a server doesn’t provide a type library, this method will return False.

This method is designed for type browsing only.

Example

{button ,AL(`OPAL_TYPE_OLEAUTO;OPAL_METH_OAENUMCONSTANTS;OPAL_METH_OAENUMCONSTANTVALUES;OPAL_METH_OAENUMEVENTS;OPAL_METH_OAENUMMETHODS;OPAL_METH_OAENUMOBJECTS;OPAL_METH_OAENUMPROPERTIES;OPAL_METH_OAENUMSERVERINFO;OPAL_METH_OAOPENOBJECTTYPEINFO;',0,"Defaultoverview",)} Related Topics

openTypeInfo example

The following example connects to the Corel Paradox type library and then builds and displays the dynamic array (DynArray) of Corel Paradox's properties:

```
method pushButton ( var eventInfo Event )
var
   oa oleauto
   dy DynArray[] String
endvar
   oa.openTypeInfo("Paradox.application")
   oa.enumProperties(dy)
   dy.view()
endMethod
```

registerControl procedure

Registers an OLE Automation control.

Syntax

```
registerControl ( const fileName String ) String
```

Description

registerControl auto-registers the OLE Automation control specified in *fileName*.

Example

{button ,AL(`OPAL_TYPE_OLEAUTO;OPAL_METH_OAENUMAUTOMATIONSERVERS;OPAL_METH_OAENUMCONTROLS;OPAL_METH_OAOPEN;OPAL_METH_OAUNREGISTERCONTROL;',0,"Defaultoverview",)} **Related Topics**

registerControl example

The following example registers the MyCntl.cntl1 control. The control's registered name is the complete pathname of the file containing the control.

```
method pushButton ( var eventInfo Event )
registerControl("C:\\OCXLIB\\MYCNTL1.OCX")
endMethod
```

unregisterControl method

Unregisters an ActiveX control.

Syntax

```
unregisterControl ( const fileName String ) Logical
```

Description

unregisterControl unregisters an ActiveX control. The argument *fileName* specifies the name of the ActiveX control you want to unregister. This method returns True if the file is a valid ActiveX control; otherwise, it returns False. The ActiveX control must support the ability to unregister itself.

Example

{button ,AL(`OPAL_TYPE_OLEAUTO;OPAL_METH_OAREGISTERCONTROL;',0,"Defaultoverview",)} Related Topics

unregisterControl example

See the **registerControl** example.

version method

Returns the version number of the current OLE2 server.

Syntax

version ( ) String

Description

version returns a string containing the version number of the currently attached OLE2 server (e.g., "2.0").

Example

{button ,AL(`OPAL_TYPE_OLEAUTO;OPAL_METH_OAATTACH;OPAL_METH_OACLOSE;OPAL_METH_OAENUMAUTOMATIONSERVERS;OPAL_METH_OAREGISTERCONTROL;OPAL_METH_OAVERSION;;',0,"Defaultoverview",)} **Related Topics**

version example

The following example opens the Corel Paradox OLE Automation server and retrieves its version number.

```
method pushButton ( var eventInfo Event )
var
   oa oleauto
   v string
endvar
   oa.open("Paradox.Application")
   v = oa.version()
endMethod
```

DataTransfer type

The DataTransfer type contains methods and procedures that create, delete, import, and export data.

Methods for the DataTransfer type

appendASCIIFix
appendASCIIVar
dlgExport
dlgImport
dlgImportASCIIFix
dlgImportASCIIVar
dlgImportSpreadsheet
dlgImportTable
empty
enumSourcePageList
enumSourceRangeList
exportASCIIFix
exportASCIIVar
exportParadoxDOS
exportSpreadsheet
getAppend
getDestCharSet
getDestDelimitedFields
getDestDelimiter
getDestFieldNamesFromFirst
getDestName
getDestSeparator
getDestType
getKeyviol
getProblems
getSourceCharSet
getSourceDelimitedFields
getSourceDelimiter
getSourceFieldNamesFromFirst
getSourceName
getSourceRange
getSourceSeparator
getSourceType
importASCIIFix
importASCIIVar
importSpreadsheet
loadDestSpec
loadSourceSpec
setAppend
setDest
setDestCharSet
setDestDelimitedFields
setDestDelimiter
setDestFieldNamesFromFirst
setDestSeparator
setKeyviol
setProblems
setSource
setSourceCharSet
setSourceDelimitedFields
setSourceDelimiter

setSourceFieldNamesFromFirst
setSourceRange
setSourceSeparator
transferData

Print related ObjectPAL methods and examples

appendASCIIFix procedure

Adds fixed format ASCII data from a file to a table.

Syntax

```
appendASCIIFix ( const fileName String, const tableName String, const specTableName String [ , const ANSI Logical ] ) Logical
```

Description

appendASCIIFix adds data from the fixed format ASCII file specified by *fileName* to the table specified by *tableName*. This method uses the layout specified in *specTableName*.

The following table illustrates the structure of the file specified with *specTableName* :

Field name	Type & size	Description
Field Name	A 25	Name of the field to import
Type	A 4	Field type to import. The Type must be a valid Corel Paradox or dBASE field specification.
Start	S	Column number where the field value begins
Length	S	Field size

This method is part of the Data Transfer type, but in previous versions it was included in the System type.

Example

{button ,AL(`OPAL_TYPE_DATATRANSFER;OPAL_METH_DXAPPENDASCIIVAR;OPAL_METH_DXEXPORTASCIIFIX;OPAL_METH_DXEXPORTASCIIVAR;OPAL_METH_DXIMPORTASCIIFIX;OPAL_METH_DXIMPORTASCIIVAR;',0,"Defaultoverview",)} Related Topics

appendASCIIFix example

The following example imports ASCII fixed text to Corel Paradox (short form):

```
ImportASCIIFix("NewRecords.txt", "TimeCards.db", "ImpSpec.db")
```

appendASCIIVar procedure

Adds delimited ASCII data from a file to a table.

Syntax

```
appendASCIIVar ( const fileName String, const tableName String [ , const separator String,
const delimiter String, const allFieldsDelimited Logical, const ANSI Logical ] ) Logical
```

Description

appendASCIIVar appends data from the delimited ASCII file specified by *fileName* to the table specified by *tableName*. **appendASCIIVar** uses the options specified by *separator*, *delimiter*, *allFieldsDelimited*, and *ANSI*.

This method is part of the Data Transfer type, but in previous versions it was included in the System type.

Example

{button ,AL(`OPAL_TYPE_DATATRANSFER;OPAL_METH_DXAPPENDASCIIFIX;OPAL_METH_DXIMPORTASCIIVAR;OPAL_METH_DXEXPORTASCIIVAR;OPAL_METH_DXIMPORTASCIIFIX;OPAL_METH_DXEXPORTASCIIFIX;',0,"Defaultoverview",)} Related Topics

appendASCIIVar example

The following example imports ASCII Delimited Text to Corel Paradox:

```
ImportASCIIVar("NewRecords.txt", "TimeCards.db")
```

dlgExport procedure

Displays the Export <*tableName*> As dialog box.

Syntax

dlgExport (const ***tableName*** String [, const ***fileName*** String] **)**

Description

dlgExport displays the Export <*tableName*> As dialog box with the specified *tableName* displayed as the default. *tableName* represents the name of the table you want to export and *fileName* specifies the name and type of the file created by the export.

ObjectPAL code suspends execution until the user closes the Export <*tableName*> As dialog box. ObjectPAL has no control over this dialog box once it is displayed; it is up to the user to close the dialog box.

Example

{button ,AL(`OPAL_TYPE_DATATRANSFER;OPAL_METH_DXDLGIMPORTASCIIFIX;OPAL_METH_DXDLGIMPORTASCIIVAR;OPAL_METH_DXDLGIMPORTSPREADSHEET;OPAL_METH_DXEXPORTASCIIFIX;OPAL_METH_DXEXPORTASCIIVAR;OPAL_METH_DXEXPORTSPREADSHEET;',0,"Defaultoverview",)} Related Topics

dlgExport example

The following example displays the Export As dialog box for the ORDERS.DB table.

```
method pushButton ( var eventInfo Event )
   var
      tableName String
   endVar


   tableName = "orders.db"
   ; invoke the Export <tablename> As dialog box
   dlgExport ( tableName )
endMethod
```

dlgImport procedure

Displays the Import Data dialog box.

Syntax

```
dlgImport ( const fileName String [ , const tableName String ] )
```

Description

dlgImport displays the Import Data dialog box with the specified file and table names displayed as the default. Corel Paradox opens text files and reads first few lines to determine whether the file contains delimited or fixed-length text. Text files are files with the *.TXT extension or with unknown extensions.

Example

{button ,AL(`OPAL_TYPE_DATATRANSFER;OPAL_METH_DXDLGEXPORT;OPAL_METH_DXDLGIMPORTASCIIFIX;OPAL_METH_DXDLGIMPORTASCIIVAR;OPAL_METH_DXDLGIMPORTSPREADSHEET;OPAL_METH_DXDLGIMPORTTABLE;',0,"Defaultoverview",)} Related Topics

dlgImport example

The following example displays the Import Data dialog box. The target table name defaults to the name of the source file, with a .DB extension. The target file type is Corel Paradox, unless another type has been specified for the table.

```
method pushButton(var eventInfo Event)
   ;the following line displays the Import Data dialog box
   dlgImport("Customer.txt")
endmethod
```

dlgImportASCIIFix procedure

Displays the Import Data dialog box.

Syntax

dlgImportASCIIFix (const ***fileName*** String **)**

Description

dlbImportASCIIFix displays the Import Data dialog box with the specified *fileName* displayed as the default, and the import file type set to fixed-length ASCII. *fileName* specifies the name of the source file and the target table for the imported data. If you specify a file extension, Corel Paradox uses it to locate the appropriate file.

The target table's extension depends on its table type. The default type for Corel Paradox is .DB; for dBASE tables the default type is .DBF. Dates and numbers are formatted according to your settings in the Windows Control Panel.

ObjectPAL code suspends execution until the user closes the Import Data dialog box.

This method is part of the Data Transfer type, but in previous versions it was included in the System type.

Example

{button ,AL(`OPAL_TYPE_DATATRANSFER;OPAL_METH_DXDLGEXPORT;OPAL_METH_DXDLGIMPORTASCIIVAR;OPAL_METH_DXDLGIMPORTSPREADSHEET;OPAL_METH_DXIMPORTASCIIFIX;',0,"Defaultoverview",)}

Related Topics

dlgImportASCIIFix example

The following example displays the Import Data dialog box and imports data from the ORDERS.TXT text file to the ORDERS.DB table:

```
method pushButton ( var eventInfo Event )
   var
      fileName String
   endVar

   fileName = "orders.txt"

   ; invoke the Import Data dialog box
   ; by default, Corel Paradox will use ORDERS.TXT as the source file
   ; and ORDERS.DB as the target table
   dlgImportASCIIFix ( fileName )
endMethod
```

dlgImportASCIIVar procedure

Displays the Import Data dialog box.

Syntax

```
dlgImportASCIIVar ( const fileName String )
```

Description

dlgImportASCIIVar displays the Import Data dialog box with the specified *fileName* displayed as the default, and the import file type set to delimited ASCII text. *fileName* specifies the name of the source file and the target table for the imported data.

The target table's extension depends on its table type. The default type for Corel Paradox is .DB; for dBASE tables the default type is .DBF. Dates and numbers are formatted according to your settings in the Windows Control Panel.

ObjectPAL code suspends execution until the user closes the Import Data dialog box.

The default settings include: fields separated by commas; fields delimited by quotes; only text fields delimited; and the OEM character set used.

This method is part of the Data Transfer type, but in previous versions it was included in the System type.

Example

{button ,AL(`OPAL_TYPE_DATATRANSFER;OPAL_METH_DXDLGEXPORT;OPAL_METH_DXDLGIMPORTASCIIFIX;OPAL_METH_DXDLGIMPORTSPREADSHEET;OPAL_METH_DXIMPORTASCIIVAR;',0,"Defaultoverview",)}

Related Topics

dlgImportASCIIVar example

The following example displays the Import Data dialog box and imports data from the ORDERS.TXT text file to the ORDERS.DB table:

```
method pushButton ( var eventInfo Event )
   var
      fileName String
   endVar

   fileName = "orders.txt"

   ; invoke the Import Data dialog box.
   ; by default, Corel Paradox will use ORDERS.TXT as the source file
   ; and ORDERS.DB as the target table.
   dlgImportASCIIVar ( fileName )
endMethod
```

dlgImportSpreadsheet procedure

Displays the Import Data dialog box.

Syntax

```
dlgImportSpreadsheet ( const fileName String )
```

Description

dlgImportSpreadsheet displays the Import Data dialog box with the specified *fileName* displayed as the default. *fileName* specifies the name of the source file, its spreadsheet type, and the name of the target table.

The target table's file extension depends on its table type. The default type for Corel Paradox is .DB; for dBASE tables the default type is .DBF. Dates and numbers are formatted according to your settings in the Windows Control Panel.

Corel Paradox uses the file extensions that you specify to identify the spreadsheet type of the source file. The following table displays the file extensions and their spreadsheet types:

Extension	Format
WB1, WB2, WB3	Quattro Pro Win
WQ1	Quattro Pro DOS
WKQ	Quattro
WK1	Lotus 2.x
WKS	Lotus 1.A
XLS	Excel 3.0/4.0/5.0

ObjectPAL code suspends execution until the user closes the Import Data dialog box.

The default settings specify that the From cell is the first cell of the spreadsheet's first page. The To cell is the last cell of the spreadsheet's last page, and the Use First Row Of Data As Field Names check box is enabled.

This method is part of the Data Transfer type, but in previous versions it was included in the System type.

Example

{button ,AL(`OPAL_TYPE_DATATRANSFER;OPAL_METH_DXDLGEXPORT;OPAL_METH_DXDLGIMPORTASCIIFIX;OPAL_METH_DXDLGIMPORTASCIIVAR;OPAL_METH_DXIMPORTSPREADSHEET;',0,"Defaultoverview",)}

Related Topics

dlgImportSpreadsheet example

The following example instructs the Import Data dialog box to import data from a Quattro Pro for Windows spreadsheet (ORDERS.WB1) to a Corel Paradox table (ORDERS.DB):

```
method pushButton ( var eventInfo Event )
   var
      fileName String
   endVar


   fileName = "orders.wb1"


   ; invoke the Import Data dialog box
   ; by default, Corel Paradox will use ORDERS.WB1 as the source file
   ; and ORDERS.DB as the target table
   dlgImportSpreadsheet ( fileName )
endMethod
```

dlgImportTable procedure

Displays the Import Data dialog box.

Syntax

```
dlgImportTable ( const tableName String )
```

Description

dlgImportTable displays the Import Data dialog box with the specified *tableName* displayed as the import source.

Example

{button ,AL(`OPAL_TYPE_DATATRANSFER;OPAL_METH_DXDLGEXPORT;OPAL_METH_DXDLGIMPORTASCIIFIX;OPAL_METH_DXDLGIMPORTASCIIVAR;OPAL_METH_DXDLGIMPORT;OPAL_METH_DXDLGIMPORTSPREADSHEET;',0,"Defaultoverview",)} Related Topics

dlgImportTable example

The following example displays the Import Data dialog box and imports data from the dBASE table ORDERS.DBF to the Corel Paradox table ORDERS.DB.

```
method pushButton ( var eventInfo Event )
   var
      tblName String
   endVar

   tblName = "orders.dbf"

   ; invoke the Import Data dialog box
   ; by default, Corel Paradox will use ORDERS.DBF as the source file
   ; and ORDERS.DB as the target table
   dlgImportTable ( tblName )
EndMethod
```

empty method

Deletes the data from a structure.

Syntax

```
empty ( )
```

Description

empty re-initializes a structure by deleting its data while leaving its form intact. **empty** can initialize Mail variable structures and tables but cannot initialize forms, databases, or reports.

Example

{button ,AL(`OPAL_TYPE_DATATRANSFER;OPAL_METH_DXDLGIMPORTSPREADSHEET;OPAL_METH_DXIMPORTSPREADSHEET;OPAL_METH_DXIMPORTASCIIFIX;OPAL_METH_DXIMPORTASCIIVAR;',0,"Defaultoverview",)} Related Topics

empty example

The following example specifies a DataTransfer data type. This structure is used with the **transferData** method. This example assumes that the DataTransfer variable, dt, is declared within a Var ... EndVar statement. The custom method **cmTransfer()** is within the scope of the variable (dt).

```
method cmTransfer()   ;this example completes a DataTransfer


   dt.setSource("CUSTOMER.TXT", DTASCIIVar)  ; sets the datatransfer source
                                             ; to CUSTOMER.TXT
   dt.setSourceSeparator("/")   ; specifies the forward slash "/" character
                                ; to separate each field
   dt.setSourceDelimiter("'")   ; specifies the single quote to surround
                                ; the fields
   dt.setSourceDelimitedFields(DTDelimJustText)  ; specifies that the single
                                                 ; quote (delimiter) surrounds
                                                 ; only text fields of the
                                                 ; source file
   dt.setSourceCharSet(DTANSI)   ; specifies that the character set used
                                 ; when creating the source file
                                 ; was the ANSI character set


   dt.setSourceFieldNamesFromFirst(False)   ; specifies to use the first
                                            ; row of the source file as
                                            ; field names
   dt.setDest("NEWCUST.DB")   ; sets the destination file to NEWCUST.DB
   dt.setProblems(True)   ; specifies to create a PROBLEMS.DB if there are
                          ; any problems importing the source file
   dt.transferData()   ; executes the data transfer. In this case it
                       ; imports the CUSTOMER.TXT file as NEWCUST.DB.
   dt.empty()   ; empties the dt variable structure to set it up for
                ; a new transfer.

endmethod
```

enumSourcePageList method

Copies the list of spreadsheet pages in a string array.

Syntax

```
enumSourcePageList ( var pages Array[] String )
```

Description

enumSourcePageList compiles a list of pages and copies it into a string array called *pages.* This method requires you to set filenames and types to an existing spreadsheet. **enumSourcePageList** only applies when the source file is a spreadsheet.

Example

{button ,AL(`OPAL_TYPE_DATATRANSFER;OPAL_METH_DXDLGIMPORTSPREADSHEET;OPAL_METH_DXIMPORTSPREADSHEET;',0,"Defaultoverview",)} Related Topics

enumSourcePageList example

The following example copies the pages of the LEDGER.WB3 spreadsheet into an array and displays the results:

```
method pushButton(var eventInfo Event)
   var
      dt      DataTransfer
      arPage  Array[]  String
   endVar
   dt.setSource("ledger.wb3")
   dt.enumSourcePageList(arPage)
   arPage.view()
endMethod
```

enumSourceRangeList method

Compiles a list of named ranges into a string array.

Syntax

```
enumSourceRangeList ( var ranges Array[] String )
```

Description

enumSourceRangeList copies the list of named ranges into a string array named *ranges.* This method requires you to set filenames and types to an existing spreadsheet. **enumSourceRangeList** only applies when the source file is a spreadsheet.

Example

{button ,AL(`OPAL_TYPE_DATATRANSFER;OPAL_METH_DXGETDESTCHARSET;OPAL_METH_DXGETSOURCERANGE;OPAL_METH_DXSETDESTCHARSET;OPAL_METH_DXSETSOURCERANGE;',0,"Defaultoverview",)}

Related Topics

enumSourceRangeList example

The following example compiles a list of the pages of the LEDGER.WB3 spreadsheet, and displays the list.

```
method pushButton(var eventInfo Event)
   var
      dt       DataTransfer
      arRange  Array[]  String
   endVar
   dt.setSource("ledger.wb3")
   dt.enumSourceRangeList(arRange)
   arRange.view()
endMethod
```

exportASCIIFix procedure

Exports data from the specified table to an ASCII file in which all of the fields are the same length.

Syntax

```
exportASCIIFix ( const tableName String, const fileName String, const specTableName String [ , const ANSI Logical ] ) Logical
```

Description

exportASCIIFix exports data from the specified table to an ASCII file in which all of the record's fields are the same length. This method duplicates the function of the Export Data dialog box.

tableName specifies the source table and *fileName* specifies the target file. If the target file does not exist, this procedure creates it using the layout specified *specTableName*. *specTableName* is the name of a table that specifies the layout for the imported data. The following table illustrates the structure of the file specified with *specTableName* :

Field name	Type & size	Description
Field Name	A 25	Name of the field to import
Type	A 4	Field type to import. The Type must be a valid Corel Paradox or dBASE field specification.
Start	S	Column number where the field value begins
Length	S	Field size

exportASCIIFix can use the same *specTableName* as **importASCIIFix**. For export operations, the table type determines the field type. More recent versions of Corel Paradox will recognize tables made with versions 5.0 and earlier, but the reverse is not true.

For each field you export, *specTableName* contains a Start position (the column where the field value begins) and a Length (the number of characters in the field). *specTableName* operates like EXPORT.DB, which is created when you use the Export Data dialog box to export a table interactively.

ANSI (optional) specifies whether to use the ANSI or OEM character set. Set *ANSI* to True to use the ANSI character set, or to False to use the OEM character set.

This method is part of the Data Transfer type, but in previous versions it was included in the System type.

Example

{button ,AL(`OPAL_TYPE_DATATRANSFER;OPAL_METH_DXDLGEXPORT;OPAL_METH_DXIMPORTASCIIFIX;OPAL_METH_DXEXPORTASCIIVAR;OPAL_METH_DXEXPORTSPREADSHEET;',0,"Defaultoverview",)} Related Topics

exportASCIIFix example

The following example exports data from the ORDERS.DB table to the ORDERS.TXT text file. The code then reads the export format from the ORDEREXP.DB table and exports the data using the ANSI character set.

```
method pushButton ( var eventInfo Event )
   exportASCIIFix ( "orders.db", "orders.txt", "orderexp.db", True )
endMethod
```

exportASCIIVar procedure

Exports data from a specified table to a delimited ASCII file. A delimited ASCII text files is one of variable fixed length.

Syntax

```
exportASCIIVar ( const tableName String, const fileName String [ , const separator String,
const delimiter String, const allFieldsDelimited Logical, const ANSI Logical ] ) Logical
```

Description

exportASCIIVar exports data from a table to a delimited ASCII file. If the file does not exist, **exportASCIIVar** creates it. This method duplicates the function of the Export Data dialog box.

tableName specifies the source table, and *fileName* specifies the target file. *separator* (optional) specifies the character that surrounds field values in the target file. You can choose a comma or any other single character, including special characters. *delimiter* (optional) specifies the character that defines the limits of field values in the target. Leave the delimiter string empty if you do not want to define limits. The *allFieldsDelimited* (optional) string is marked True if data from all field types is delimited, and False if data from only text, alphanumeric, or character field types is delimited.

Corel Paradox cannot export memo (Corel Paradox or dBASE), formatted memo, graphic, OLE, or binary fields to delimited text.

ANSI (optional) specifies whether to use the ANSI or OEM character set. Set *ANSI* to True to use the ANSI character set, or to False to use the OEM character set.

The following table displays the default settings for optional arguments:

separator	"," (comma)
delimiter	"\"" (double quote)
allFieldsDelimited	False
ANSI	False

This method is part of the Data Transfer type, but in previous versions it was included in the System type.

Example

{button ,AL(`OPAL_TYPE_DATATRANSFER;OPAL_METH_DXDLGEXPORT;OPAL_METH_DXEXPORTASCIIFIX;OPAL_METH_DXEXPORTSPREADSHEET;',0,"Defaultoverview",)} Related Topics

exportASCIIVar example

The following example exports data from the ORDERS.DB table to the ORDERS.TXT text file. In this example, tabs delimit field values, percent signs enclose each value, only text fields are delimited, and the ANSI character set is used.

```
method pushButton ( var eventInfo Event )
   exportASCIIVar ( "orders.db", "orders.txt", "\t", "%", False, True )
endMethod
```

The following code exports Corel Paradox to ASCII Delimited Text (medium form):

```
var
   dt DataTransfer
endVar


dt.setSource("TimeCards.db")
dt.setDest("Records.txt", DTAsciiVar)
dt.TransferData()
```

The following code exports Corel Paradox to ASCII Delimited Text (short form):

```
ExportASCIIVar("TimeCards.db", "NewRecords.txt")
```

exportParadoxDOS procedure

Exports data from a Corel Paradox for Windows or a dBASE table to a Level 4 Corel Paradox for DOS table.

Syntax

```
exportParadoxDOS ( const tableName String, const fileName String ) Logical
```

Description

exportParadoxDOS exports data from a Corel Paradox for Windows or a dBASE table to a Level 4 Corel Paradox for DOS table. This method duplicates the function of the Table Export dialog box.

exportParadoxDOS cannot export Bytes (type Y) fields because they are excluded from the destination file. This method does not export OLE and Binary fields when you export a dBASE table to Corel Paradox for DOS format.

tableName specifies the source table and *fileName* specifies the target file. If you include a file extension with the filename, it must be *.DB.

This method is part of the Data Transfer type, but in previous versions it was included in the System type.

Example

{button ,AL(`OPAL_TYPE_DATATRANSFER;OPAL_METH_DXDLGEXPORT;OPAL_METH_DXEXPORTASCIIFIX;OPAL_METH_DXEXPORTASCIIVAR;OPAL_METH_DXEXPORTSPREADSHEET;',0,"Defaultoverview",)} Related Topics

exportParadoxDOS example

The following example exports data from a dBASE table named ORDERS.DBF to a Corel Paradox for DOS table named ORDERS.DB:

```
method pushButton ( var eventInfo Event )
   if not exportParadoxDOS ( "orders.dbf", "orders" ) then
      errorShow ( "Export to Corel Paradox DOS failed." )
   endIf
endMethod
```

exportSpreadsheet procedure

Exports data from a table to a spreadsheet file.

Syntax

```
exportSpreadsheet ( const tableName String, const fileName String [ , const makeRowHeaders Logical ] ) Logical
```

Description

exportSpreadsheet exports the data from a table to a spreadsheet file, duplicating the function of the Export Data dialog box. If the spreadsheet file does not exist, this method creates it. The spreadsheet type is determined by the file extension. When you export data to a spreadsheet, Corel Paradox converts each record to a row and each field to a column. If a value is wider than the column, the full value is partially hidden.

This method is part of the Data Transfer type, but in previous versions it was included in the System type.

If a date in the original table is beyond the acceptable range in the spreadsheet, the date is exported as an ERROR.

tableName specifies the source table and *fileName* specifies the target file. *makeRowHeaders* (optional) specifies whether the table's column headers correspond to the spreadsheet's rows The *makeRowHeaders* string returns True (default) if column headers are used as labels, and False if they are not.

The file extension in *fileName* specifies the format of the spreadsheet file. The following table displays file extensions and their spreadsheet formats:

Extension	Format
WB1, WB2, WB3	Quattro Pro Win
WQ1	Quattro Pro DOS
WKQ	Quattro
WK1	Lotus 2.x
WKS	Lotus 1.A
XLS	Excel 3.0/4.0/5.0

Example

{button ,AL(`OPAL_TYPE_DATATRANSFER;OPAL_METH_DXDLGEXPORT;OPAL_METH_DXEXPORTASCIIFIX;OPAL_METH_DXEXPORTASCIIVAR;',0,"Defaultoverview",)} Related Topics

exportSpreadsheet example

The following example exports data from the ORDERS.DB table to a Quattro Pro for Windows file. The table's field names are used as labels in the spreadsheet file.

```
method pushButton ( var eventInfo Event )
   exportSpreadsheet ( "orders.db", "orders.wb1", True )
endMethod
```

getAppend method

Retrieves the True or False value set by **setAppend**.

Syntax

```
getAppend ( ) Logical
```

Description

getAppend retrieves the True or False value set by **setAppend**. **getAppend** applies only when the destination is a table.

Example

{button ,AL(`OPAL_TYPE_DATATRANSFER;OPAL_METH_DXAPPENDASCIIFIX;OPAL_METH_DXAPPENDASCIIVAR;OPAL_METH_DXDLGIMPORTASCIIFIX;OPAL_METH_DXDLGIMPORTASCIIVAR;OPAL_METH_DXDLGIMPORT;OPAL_METH_DXDLGIMPORTSPREADSHEET;OPAL_METH_DXDLGIMPORTTABLE;OPAL_METH_DXIMPORTASCIIFIX;OPAL_METH_DXIMPORTASCIIVAR;OPAL_METH_DXIMPORTSPREADSHEET;OPAL_METH_DXSETAPPEND;',0,"Defaultoverview",)} Related Topics

getAppend example

The following example specifies a DataTransfer data type. Use this code to build an Import or Export specification, and then call the **transferData** method.

```
var
   dt DataTransfer
endVar
dt.SetSource ( "MYFILE.TXT" )
if dt.getSourceType ( ) = DTASCIIFixed Then
   dt.loadDestSpec ( "SpecTable" )
EndIf
dt.setDest ( "Existing Data.db" )
if not dt.getAppend () then
   dt.setAppend ( True )
endif
dt.transferData ( )
```

getDestCharSet method

Retrieves the value set by **setDestCharSet**.

Syntax

```
getDestCharSet ( ) SmallInt
```

Description

getSourceCharSet retrieves the value set by **setDestCharSet**. The value is one of the two DataTransferCharset constants: dtOEM or dtANSI. **getSourceCharSet** applies only when the source or destination is a fixed or delimited ASCII text file.

Example

{button ,AL(`OPAL_TYPE_DATATRANSFER;OPAL_METH_DXENUMSOURCERANGELIST;OPAL_METH_DXGETDESTDELIMITEDFIELDS;OPAL_METH_DXGETDESTDELIMITER;OPAL_METH_DXGETDESTFIELDNAMESFROMFIRST;OPAL_METH_DXGETDESTNAME;OPAL_METH_DXGETDESTSEPARATOR;OPAL_METH_DXGETDESTTYPE;OPAL_METH_DXGETSOURCERANGE;OPAL_METH_DXSETDESTCHARSET;OPAL_METH_DXSETSOURCERANGE;',0,"Defaultoverview",)} Related Topics

getDestCharSet example

The following example uses the **transferData** method to export ORDERS.DB to ORDINFO.TXT. It uses **setDestCharSet** to specify the use of the ANSI character set. To specify the use of the OEM character set, use the DTOEM constant with **setDestCharSet**.

```
method pushButton(var eventInfo Event)
   var
      dt      DataTransfer
   endVar
   dt.setDest("ordinfo.txt", DTASCIIVar)
   dt.setSource("orders.db")


   ;Specify the single quote (') to surround the fields.
   ;The delimited fields will be text fields only.
   dt.setDestDelimiter("'")
   dt.setDestDelimitedFields(DTDelimJustText)


   ;Specify the tab character to separate the fields.
   dt.setDestSeparator("\t")


   ;Set the first row of the ORDINFO.TXT to be the field names
   dt.setDestFieldNamesFromFirst(True)


   ;Set the character set of the destination file ORDINFO.TXT to be the ANSI
   ;character set.
   if dt.getDestCharSet()= dtOEM then
      dt.setDestCharSet(DTAnsi)
   endif
   ;run Export
   dt.transferData()
endMethod
```

getDestDelimitedFields method

Retrieves the value set by **setDestDelimitedFields**.

Syntax

```
getDestDelimitedFields ( ) SmallInt
```

Description

getDestDelimitedFields retrieves the value set by the **setDestDelimitedFields** method. The value is one of the two DataTransferDelimitCode constants: DtDelimAllFields or DtDelimJustText. **getDestDelimitedFields** only applies when the source or the destination is a delimited ASCII text file.

Example

{button ,AL(`OPAL_TYPE_DATATRANSFER;OPAL_METH_DXGETDESTCHARSET;OPAL_METH_DXGETDESTDELIMITER;OPAL_METH_DXGETDESTFIELDNAMESFROMFIRST;OPAL_METH_DXGETDESTNAME;OPAL_METH_DXGETDESTSEPARATOR;OPAL_METH_DXGETDESTTYPE;OPAL_METH_DXSETDESTDELIMITEDFIELDS;OPAL_METH_DXSETDESTDELIMITER;OPAL_METH_DXSETDESTSEPARATOR;',0,"Defaultoverview",)} Related Topics

getDestDelimitedFields example

The following example uses the **transferData** method to export ORDERS.DB to ORDINFO.TXT. It uses **setDestDelimitedFields** to delimit surrounding text fields only. To delimit all fields, use the DTDelimAllFields constant with **setDestDelimitedFields**.

```
method pushButton(var eventInfo Event)
   var
      dt      DataTransfer
   endVar
   dt.setDest("ordinfo.txt", DTASCIIVar)
   dt.setSource("orders.db")


   ;Specify the single quote (') to surround the fields.
   ;The delimited fields will be text fields only.
   dt.setDestDelimiter("'")
   dt.setDestDelimitedFields(DTDelimJustText)


   ;Specify the tab character to separate the fields.
   dt.setDestSeparator("\t")


   ;Set the first row of the ORDINFO.TXT to be the field names
   dt.setDestFieldNamesFromFirst(True)


   ;Set the character set of the destination file ORDINFO.TXT to be the ANSI
   ;character set.
   dt.setDestCharSet(DTAnsi)


   ;run Export
   dt.transferData()
endMethod
```

getDestDelimiter method

Retrieves the value set by **setDestDelimiter**.

Syntax

```
getDestDelimiter ( ) String
```

Description

getDestDelimiter retrieves the value set by the **setDestDelimiter** method. **getDestDelimiter** only applies when the source or the destination is a delimited ASCII text file.

Example

{button ,AL(`OPAL_TYPE_DATATRANSFER;OPAL_METH_DXGETDESTCHARSET;OPAL_METH_DXGETDESTDELIMITEDFIELDS;OPAL_METH_DXGETDESTFIELDNAMESFROMFIRST;OPAL_METH_DXGETDESTNAME;OPAL_METH_DXGETDESTSEPARATOR;OPAL_METH_DXGETDESTTYPE;OPAL_METH_DXSETDESTDELIMITEDFIELDS;OPAL_METH_DXSETDESTDELIMITER;OPAL_METH_DXSETDESTSEPARATOR;',0,"Defaultoverview",)}

Related Topics

getDestDelimiter example

The following example exports the ORDERS.DB table into an ASCII delimited text file. The single quote character is specified as a delimiter.

```
method pushButton(var eventInfo Event)
   var
      dt      DataTransfer
   endVar
   dt.setDest("ordinfo.txt", DTASCIIVar)
   dt.setSource("orders.db")


   ;Specify the single quote (') to surround the fields.
   ;The delimited fields will be text fields only.
   msgInfo("Info", "current delimiter is "+dt.getDestDelimiter())
   dt.setDestDelimiter("'")
   dt.setDestDelimitedFields(DTDelimJustText)


   ;Specify the tab character to separate the fields.
   dt.setDestSeparator("\t")


   ;Set the first row of the ORDINFO.TXT to be the field names
   dt.setDestFieldNamesFromFirst(True)


   ;Set the character set of the destination file ORDINFO.TXT to be the ANSI
   ;character set.
   dt.setDestCharSet(DTAnsi)


   ;run Export
   dt.transferData()
endMethod
```

getDestFieldNamesFromFirst method

Retrieves the True or False value set by **setDestFieldNamesFromFirst**.

Syntax

```
getDestFieldNamesFromFirst ( ) Logical
```

Description

getDestFieldNamesFromFirst retrieves the True or False value set by **setDestFieldNamesFromFirst**.
getDestFieldNamesFromFirst only applies when the source file is a spreadsheet.

Example

{button ,AL(`OPAL_TYPE_DATATRANSFER;OPAL_METH_DXGETDESTCHARSET;OPAL_METH_DXGETDESTDELIMITEDFIELDS;OPAL_METH_DXGETDESTDELIMITER;OPAL_METH_DXGETDESTNAME;OPAL_METH_DXGETDESTSEPARATOR;OPAL_METH_DXGETDESTTYPE;OPAL_METH_DXSETDESTFIELDNAMESFROMFIRST;',0,"Defaultoverview",)} **Related Topics**

getDestFieldNamesFromFirst example

The following example uses the **transferData** method to export ORDERS.DB to ORDINFO.TXT. The **setDestFieldNamesFromFirst** is used to create the first row of the text file with field names.

```
method pushButton(var eventInfo Event)
   var
      dt      DataTransfer
   endVar
   dt.setDest("ordinfo.txt", DTASCIIVar)
   dt.setSource("orders.db")


   ;Specify the single quote (') to surround the fields.
   ;The delimited fields will be text fields only.
   dt.setDestDelimiter("'")
   dt.setDestDelimitedFields(DTDelimJustText)


   ;Specify the tab character to separate the fields.
   dt.setDestSeparator("\t")


   If dt.getDestFieldNamesFromFirst() Then
      msgInfo("Info", "SetDestFieldNamesFromFirst is On")
   else
      msgInfo("Info", "Setting DestFieldNamesFrom First to On")
   endIf
;Set the first row of the ORDINFO.TXT to be the field names
   dt.setDestFieldNamesFromFirst(True)


   ;Set the character set of the destination file ORDINFO.TXT to be the ANSI
   ;character set.
   dt.setDestCharSet(DTAnsi)


   ;run Export
   dt.transferData()
endMethod
```

getDestName method

Retrieves the destination filename.

Syntax

```
getDestName ( ) String
```

Description

getDestName retrieves the destination filename.

Example

{button ,AL(`OPAL_TYPE_DATATRANSFER;OPAL_METH_DXGETDESTCHARSET;OPAL_METH_DXGETDESTDELIMITEDFIELDS;OPAL_METH_DXGETDESTDELIMITER;OPAL_METH_DXGETDESTFIELDNAMESFROMFIRST;OPAL_METH_DXGETDESTSEPARATOR;OPAL_METH_DXGETDESTTYPE;OPAL_METH_DXGETSOURCENAME;OPAL_METH_DXGETSOURCETYPE;OPAL_METH_DXSETDEST;OPAL_METH_DXSETSOURCE;',0,"Defaultoverview",)} Related Topics

getDestName example

The following example specifies a DataTransfer data type. Use this code to build an Import or Export specification and then call the **transferData** method.

Export to text

```
var
   dt DataTransfer
endVar
dt.setSource ( "ANSWER.db" )
msgInfo("Info", "The current source is " + dt.getSourceName())
dt.SetDest ( "NEWFILE.TXT",dtASCIIVar)
msgInfo("Info", "The current destination is " + dt.getDestName())
dt.setDestSeparator ( ";" )
dt.transferData ( )
```

getDestSeparator method

Retrieves the value set by **setDestSeparator**.

Syntax

```
getDestSeparator ( ) String
```

Description

getDestSeparator retrieves the value set by the **setDestSeparator** method. **getDestSeparator** only applies when the source or the destination is a delimited ASCII text file.

Example

{button ,AL(`OPAL_TYPE_DATATRANSFER;OPAL_METH_DXGETDESTCHARSET;OPAL_METH_DXGETDESTDELIMITEDFIELDS;OPAL_METH_DXGETDESTDELIMITER;OPAL_METH_DXGETDESTFIELDNAMESFROMFIRST;OPAL_METH_DXGETDESTNAME;OPAL_METH_DXGETDESTSEPARATOR;OPAL_METH_DXGETDESTTYPE;OPAL_METH_DXSETDESTDELIMITEDFIELDS;OPAL_METH_DXSETDESTDELIMITER;',0,"Defaultoverview",)}

Related Topics

getDestSeparator example

The following examples specify a DataTransfer data type. Use this code to build an Import or Export specification and then call the **transferData** method.

Import from spreadsheet

```
var
   dt DataTransfer
endVar
; Source file MYFILE.WB2 is a Quattro Pro Windows spreadsheet

dt.setSource ( "MYFILE.WB2" )
msgInfo("Info", "The current source separator is " + dt.getSourceSeparator())
; Destination file is a Corel Paradox table NEWFILE.DB

dt.SetDest ( "NEWFILE.DB" )
msgInfo("Info", "The current destination separator is " + dt.getDestSeparator())
dt.setProblems ( True )
dt.transferData ( )
```

Import from text

```
var
   dt DataTransfer
endVar
msgInfo("Info", "The current source separator is " + dt.getSourceSeparator())
dt.setSource ( "SRCFILE.TXT")

MsgInfo("Info", "The current destination separator is " + dt.getDestSeparator())
dt.SetDest ( "NEWFILE.db" )
if dt.getSourceType ( ) = DTASCIIFixed Then
   dt.loadDestSpec ( "SpecTable" )
EndIf
dt.setAppend ( True )
dt.transferData ( )
```

Export to text

```
var
   dt DataTransfer
endVar
msgInfo("Info", "The current source separator is " + dt.getSourceSeparator())
dt.setSource ( "CUSTOMER.db" )
msgInfo("Info", "The current destination separator is " + dt.getDestSeparator())
dt.SetDest ( "NEWFILE.TXT",dtASCIIVar )
dt.setDestSeparator ( ";" )
dt.transferData ( )
```

getDestType method

Retrieves the destination file type constant.

Syntax

```
getDestType ( ) SmallInt
```

Description

getDestType retrieves the destination file type constant. The file type constant is one of the DataTransferFileType constants.

Example

{button ,AL(`OPAL_TYPE_DATATRANSFER;OPAL_METH_DXGETDESTCHARSET;OPAL_METH_DXGETDESTDELIMITEDFIELDS;OPAL_METH_DXGETDESTDELIMITER;OPAL_METH_DXGETDESTFIELDNAMESFROMFIRST;OPAL_METH_DXGETDESTNAME;OPAL_METH_DXGETDESTSEPARATOR;OPAL_METH_DXGETSOURCENAME;OPAL_METH_DXGETSOURCETYPE;OPAL_METH_DXSETDEST;OPAL_METH_DXSETSOURCE;',0,"Defaultoverview",)} Related Topics

getDestType example

The following example specifies a DataTransfer data type. Use this code to build an Import or Export specification and then call the **transferData** method.

Import from spreadsheet

```
var
   dt DataTransfer
endVar
msgInfo("Info", "the current dest type is " + string(dt.getDestType()))
dt.setSource ( "MYFILE.WKS" )
dt.setDest ( "New Data.db" )
dt.setProblems ( True )
dt.transferData ( )
```

Import from text

```
var
   dt DataTransfer
endVar
dt.SetSource ( "MYFILE.TXT" )
if dt.getSourceType ( ) = DTASCIIFixed Then
   dt.loadDestSpec ( "SpecTable" )
EndIf
msgInfo("Info", "the current dest type is " + string(dt.getDestType()))
dt.setDest ( "Existing Data.db" )
dt.setAppend ( True )
dt.transferData ( )
```

Export to text

```
var
   dt DataTransfer
endVar
dt.setSource ( "ANSWER.db" )
msgInfo("Info", "the current dest type is " + string(dt.getDestType()))
dt.SetDest ( "NEWFILE.TXT" )
dt.setDestSeparator ( ";" )
dt.transferData ( )
```

getKeyviol method

Retrieves the True or False value set by **setKeyviol**.

Syntax

```
getKeyviol ( [ const tableName String, var count LongInt ] ) Logical
```

Description

getKeyviol retrieves the True or False value set by the **setKeyviol** method. The argument *tableName* is the name of the Key Violations table, and *count* is the number of key violations in the table. **getKeyviol** method applies only when the destination is a table.

Example

{button ,AL(`OPAL_TYPE_DATATRANSFER;OPAL_METH_DXSETKEYVIOL;OPAL_METH_DXSETPROBLEMS;OPAL_METH_DXGETPROBLEMS;',0,"Defaultoverview",)} Related Topics

getKeyviol example

The following example retrieves the key violations from the kvTbl file.

```
method pushButton(var eventInfo Event)
   var
      dt                  DataTransfer
      kvTbl, probTbl      String
      kvNum, probNum      Longint
   endVar
   dt.setSource("MYFILE.TXT")
   dt.LoadSourceSpec("SPECFILE.DB")
   dt.setDest("MYFILE.DB")
   if isTable("MYFILE.DB ") then
      dt.setAppend(True)
   endIf
   if msgQuestion("Import Option",
                  "Would you like to produce auxilliary tables?") = "Yes" then
      dt.setKeyviol(True)
      dt.setProblems(True)
   endIF
   dt.transferData()
   if dt.getKeyviol(kvTbl, kvNum) then
      msgInfo("Import Status",
            "# Key violations = "+string(kvNum)+
            "\nKeyviol table name = " + kvTbl)
   endIf
   if dt.getProblems(probTbl, probNum) then
         msgInfo("Import Status",
               "# Record errors = "+string(probNum) +
               "\nProblem table name = " + probTbl)
   endIf
endMethod
```

getProblems method

Retrieves the True or False value set by **setProblems**.

Syntax

```
getProblems ( [ var tableName String, var count LongInt ] ) Logical
```

Description

getProblems retrieves the True or False value set by the **setProblems** method. The *tableName* argument specifies the name of the problems table, and *count* specifies number of problems. **getProblems** applies only when the destination is a table.

Example

{button ,AL(`OPAL_TYPE_DATATRANSFER;OPAL_METH_DXSETKEYVIOL;OPAL_METH_DXSETPROBLEMS;OPAL_METH_DXGETKEYVIOL;',0,"Defaultoverview",)} Related Topics

getProblems example

The following example retrieves the problems from the probTbl file.

```
method pushButton(var eventInfo Event)
   var
      dt                  DataTransfer
      kvTbl, probTbl      String
      kvNum, probNum      Longint
   endVar
   dt.setSource("MYFILE.TXT")
   dt.LoadSourceSpec("SPECFILE.DB")
   dt.setDest("MYFILE.DB")
   if isTable("MYFILE.DB ") then
      dt.setAppend(True)
   endIf
   if msgQuestion("Import Option",
                  "Would you like to produce auxilliary tables?") = "Yes" then
      dt.setKeyviol(True)
      dt.setProblems(True)
   endIF
   dt.transferData()
   if dt.getKeyviol(kvTbl, kvNum) then
      msgInfo("Import Status",
            "# Key violations = "+string(kvNum)+
            "\nKeyviol table name = " + kvTbl)
   endIf
   if dt.getProblems(probTbl, probNum) then
         msgInfo("Import Status",
               "# Record errors = "+string(probNum) +
               "\nProblem table name = " + probTbl)
   endIf
endMethod
```

getSourceCharSet method

Retrieves the value set by **setSourceCharSet**.

Syntax

```
getSourceCharSet ( ) SmallInt
```

Description

getSourceCharSet retrieves the value set by **setSourceCharSet**. The value is one of the two DataTransferCharset constants: dtOEM or dtANSI. **getSourceCharSet** applies only when the source or destination is a fixed or delimited ASCII text file.

Example

{button ,AL(`OPAL_TYPE_DATATRANSFER;OPAL_METH_DXENUMSOURCERANGELIST;OPAL_METH_DXGETSOURCEDELIMITEDFIELDS;OPAL_METH_DXGETSOURCEDELIMITER;OPAL_METH_DXGETSOURCEFIELDNAMESFROMFIRST;OPAL_METH_DXGETSOURCENAME;OPAL_METH_DXGETSOURCERANGE;OPAL_METH_DXGETSOURCESEPARATOR;OPAL_METH_DXGETSOURCETYPE;OPAL_METH_DXSETSOURCECHARSET;OPAL_METH_DXSETSOURCERANGE;',0,"Defaultoverview",)} **Related Topics**

getSourceCharSet example

The following example uses the **transferData** method to export ORDERS.DB to ORDINFO.TXT. It uses **getSourceCharSet** to determine the source file's character set and **setDestCharSet** to set the destination file to the same character set. To specify the OEM character set, use the DTOEM constant with **setDestCharSet**.

```
method pushButton(var eventInfo Event)
   var
      dt      DataTransfer

      stChrSt String
   endVar
   dt.setDest("ordinfo.txt", DTASCIIVar)
   dt.setSource("orders.db")


   ;Specify the single quote (') to surround the fields.
   ;The delimited fields will be text fields only.
   dt.setDestDelimiter("'")
   dt.setDestDelimitedFields(DTDelimJustText)


   ;Specify the tab character to separate the fields.
   dt.setDestSeparator("\t")


   ;Set the first row of the ORDINFO.TXT to be the field names
   dt.setDestFieldNamesFromFirst(True)


   ; set the destination character set to be the same as the source.

   if dt.getSourceCharSet() = dtAnsi then

        dt.setDestCharSet(DTAnsi)
        stChrSt = "ANSI"

   else

        dt.setDestCharSet(DTOEM)
        stChrSt = "OEM"

   endif


   ; since the result of getSourceCharSet is a SmallInt, convert the result

   ; to a string which represents ANSI or OEM

   msgInfo("Info", "the character set is: " + stChrSt)

   ;run Export
   dt.transferData()
endMethod
```

getSourceDelimitedFields method

Retrieves the value set by **setSourceDelimitedFields**.

Syntax

```
getSourceDelimitedFields ( ) SmallInt
```

Description

getSourceDelimitedFields retrieves the value set by the **setSourceDelimitedFields** method. The value is one of the two DataTransferDelimitCode constants: DtDelimAllFields or DtDelimJustText.
getSourceDelimitedFields only applies when the source or the destination is a delimited ASCII text file.

Example

{button ,AL(`OPAL_TYPE_DATATRANSFER;OPAL_METH_DXGETDESTDELIMITER;OPAL_METH_DXGETDESTSEPARATOR;OPAL_METH_DXGETSOURCECHARSET;OPAL_METH_DXGETSOURCEDELIMITER;OPAL_METH_DXGETSOURCEFIELDNAMESFROMFIRST;OPAL_METH_DXGETSOURCENAME;OPAL_METH_DXGETSOURCERANGE;OPAL_METH_DXGETSOURCESEPARATOR;OPAL_METH_DXGETSOURCETYPE;OPAL_METH_DXSETDESTDELIMITEDFIELDS;OPAL_METH_DXSETDESTDELIMITER;OPAL_METH_DXSETDESTSEPARATOR;',0,"Defaultoverview",)} Related Topics

getSourceDelimitedFields example

The following example uses **getSourceDelimitedFields** to determine which fields are delimited.

```
method pushButton(var eventInfo Event)
   var
      dt      DataTransfer
   endVar
   dt.setSource("iesimpld.txt")


;The following lines check to see what Corel Paradox determined as the type of
;of the source file. If it is delimited, Corel Paradox determines the separator,
;delimiter and which fields are delimited.
   switch
      case dt.getSourceType() = DTASCIIVar :
         fldType = "Delimited"
         fldDelimiter = dt.getSourceDelimiter()
         if dt.getSourceDelimitedFields() = DTDelimAllFields then
            fldDelimitedFields = "All"
         else
            fldDelimitedFields = "Text"
         endIF
         fldSeparator = dt.getSourceSeparator()
      case dt.getSourceType() = DTASCIIFixed :
         fldType = "Fixed"
      otherwise :
         msgInfo("Hello","File missing or not text.")
   endSwitch
endMethod
```

getSourceDelimiter method

Retrieves the value set by **setSourceDelimiter**.

Syntax

```
getSourceDelimiter ( ) String
```

Description

getSourceDelimiter retrieves the value set by the **setSourceDelimiter** method. **getSourceDelimiter** only applies when the source or the destination is a delimited ASCII text file.

Example

{button ,AL(`OPAL_TYPE_DATATRANSFER;OPAL_METH_DXGETDESTDELIMITEDFIELDS;OPAL_METH_DXGETDESTSEPARATOR;OPAL_METH_DXGETSOURCECHARSET;OPAL_METH_DXGETSOURCEDELIMITEDFIELDS;OPAL_METH_DXGETSOURCEFIELDNAMESFROMFIRST;OPAL_METH_DXGETSOURCENAME;OPAL_METH_DXGETSOURCERANGE;OPAL_METH_DXGETSOURCESEPARATOR;OPAL_METH_DXGETSOURCETYPE;OPAL_METH_DXSETDESTDELIMITEDFIELDS;OPAL_METH_DXSETDESTDELIMITER;OPAL_METH_DXSETDESTSEPARATOR;',0,"Defaultoverview",)} **Related Topics**

getSourceDelimiter example

The following example uses **getSourceDelimiter** to display the delimiter used in the source file. This confirms that the delimiter is set correctly and allows you to specify a new delimiter if necessary.

```
method pushButton(var eventInfo Event)
   var
      dt      DataTransfer
   endVar
   dt.setSource("iesimpld.txt")


;The following lines check to see what Corel Paradox determined as the type of
;of the source file. If it is delimited, Corel Paradox determines the separator,
;delimiter and which fields are delimited.
   switch
      case dt.getSourceType() = DTASCIIVar :
         fldType = "Delimited"
         fldDelimiter = dt.getSourceDelimiter()
         if dt.getSourceDelimitedFields() = DTDelimAllFields then
            fldDelimitedFields = "All"
         else
            fldDelimitedFields = "Text"
         endIF
         fldSeparator = dt.getSourceSeparator()
      case dt.getSourceType() = DTASCIIFixed :
         fldType = "Fixed"
      otherwise :
         msgInfo("Hello","File missing or not text.")
   endSwitch
endMethod
```

getSourceFieldNamesFromFirst method

Retrieves the True or False value set by **setSourceFieldNamesFromFirst**.

Syntax

```
getSourceFieldNamesFromFirst ( ) Logical
```

Description

getSourceFieldNamesFromFirst retrieves the True or False value set by **setSourceFieldNamesFromFirst**. **getSourceFieldNamesFromFirst** only applies when the source is a spreadsheet.

Example

{button ,AL(`OPAL_TYPE_DATATRANSFER;OPAL_METH_DXDLGIMPORTSPREADSHEET;OPAL_METH_DXEXPORTSPREADSHEET;OPAL_METH_DXGETSOURCECHARSET;OPAL_METH_DXGETSOURCEDELIMITEDFIELDS;OPAL_METH_DXGETSOURCEDELIMITER;OPAL_METH_DXGETSOURCENAME;OPAL_METH_DXGETSOURCERANGE;OPAL_METH_DXGETSOURCESEPARATOR;OPAL_METH_DXGETSOURCETYPE;OPAL_METH_DXIMPORTSPREADSHEET;OPAL_METH_DXSETDESTFIELDNAMESFROMFIRST;',0,"Defaultoverview",)} **Related Topics**

getSourceFieldNamesFromFirst example

The following example specifies a DataTransfer data type. This structure is used with the **transferData** method. This example assumes that the DataTransfer variable (dt) is declared within a Var ... EndVar statement. The custom method **cmTransfer( )** is within the scope of the variable (dt).

```
method cmTransfer()   ;this example completes a DataTransfer


   dt.setSource("CUSTOMER.TXT", DTASCIIVar)  ; sets the datatransfer source
                                             ; to CUSTOMER.TXT
   dt.setSourceSeparator("/")   ; specifies the forward slash "/" character
                                ; to separate each field
   dt.setSourceDelimiter("'")   ; specifies the single quote to surround
                                ; the fields
   dt.setSourceDelimitedFields(DTDelimJustText)  ; specifies that the single
                                                 ; quote (delimiter) surrounds
                                                 ; only text fields of the
                                                 ; source file
   dt.setSourceCharSet(DTANSI)   ; specifies that the character set used
                                 ; when creating the source file
                                 ; was the ANSI character set
   msgInfo("Info", "the current setting is " + string(dt.getSourceFieldNamesFromFirst()))
   dt.setSourceFieldNamesFromFirst(False)   ; specifies to use the first
                                            ; row of the source file as
                                            ; field names
   dt.setDest("NEWCUST.DB")   ; sets the destination file to NEWCUST.DB
   dt.setProblems(True)   ; specifies to create a PROBLEMS.DB if there are
                          ; any problems importing the source file
   dt.transferData()   ; executes the data transfer. In this case it
                       ; imports the CUSTOMER.TXT file as NEWCUST.DB.
   dt.empty()   ; empties the dt variable structure to set it up for
                ; a new transfer.

endmethod
```

getSourceName method

Retrieves the source filename.

Syntax

```
getSourceName ( ) String
```

Description

getSourceName retrieves the source filename.

Example

{button ,AL(`OPAL_TYPE_DATATRANSFER;OPAL_METH_DXGETDESTNAME;OPAL_METH_DXGETDESTTYPE;OPAL_METH_DXGETSOURCECHARSET;OPAL_METH_DXGETSOURCEDELIMITEDFIELDS;OPAL_METH_DXGETSOURCEDELIMITER;OPAL_METH_DXGETSOURCEFIELDNAMESFROMFIRST;OPAL_METH_DXGETSOURCERANGE;OPAL_METH_DXGETSOURCESEPARATOR;OPAL_METH_DXGETSOURCETYPE;OPAL_METH_DXSETDEST;OPAL_METH_DXSETSOURCE;',0,"Defaultoverview",)} Related Topics

getSourceName example

The following example determines if the user has attempted to import data from the SYSTEM.INI file.

```
method gtSrcName()

var
   dt DataTransfer
   importSourceFile String
endVar


importSourceFile = "Your sourcename here"
importsourcefile.view("Import what file?")


dt.setSource(importSourceFile, dtAuto)  ;// allow Corel Paradox to determine filetype
if dt.getSourceName() = "system.ini" then
   msgStop("No!", "This source file won't create useable data.")
   return
else
   dt.setDest("importSample", dtParadox8)  ;// import into Corel Paradox 7 table
   dt.transferData ( )
endif
endMethod
```

getSourceRange method

Retrieves the range set by **setSourceRange**.

Syntax

```
getSourceRange ( ) String
```

Description

getSourceRange retrieves the range set by the **setSourceRange** method. **getSourceRange** only applies when the source is a spreadsheet.

Example

{button ,AL(`OPAL_TYPE_DATATRANSFER;OPAL_METH_DXENUMSOURCERANGELIST;OPAL_METH_DXGETDESTCHARSET;OPAL_METH_DXGETSOURCECHARSET;OPAL_METH_DXGETSOURCEDELIMITEDFIELDS;OPAL_METH_DXGETSOURCEDELIMITER;OPAL_METH_DXGETSOURCEFIELDNAMESFROMFIRST;OPAL_METH_DXGETSOURCENAME;OPAL_METH_DXGETSOURCESEPARATOR;OPAL_METH_DXGETSOURCETYPE;OPAL_METH_DXSETDESTCHARSET;OPAL_METH_DXSETSOURCERANGE;',0,"Defaultoverview",)} Related Topics

getSourceRange example

The following demonstrates the **setSourceRange** method. You can use this method to specify the range in a spreadsheet to import. **getSourceRange** accepts both named ranges and standard ranges.

```
method pushButton(var eventInfo Event)
   var
      dt      DataTransfer
   endVar
   dt.setSource("092595.wb2")


   ;Set the range to import from the spreadsheet.
   ;Either named range or specified range (ie. Page1:A1..Page3:AB10)
   msgInfo("Info", "The Current range is " + dt.getSourceRange())
   dt.setSourceRange("myRange")
   dt.setSourceFieldNamesFromFirst(True)
   dt.setDest("delme09.db")


   ;Prompt the user to verify range to import. getSourceRange returns the
   ;actual range notation.
   view(dt.getSourceRange(),"Import Range")
   dt.transferData()
endMethod
```

getSourceSeparator method

Retrieves the value set by **setSourceSeparator**.

Syntax

```
getSourceSeparator ( ) String
```

Description

getSourceSeparator retrieves the value set by the **setSourceSeparator** method. **getSourceSeparator** only applies when the source or the destination is a delimited ASCII text file.

Example

{button ,AL(`OPAL_TYPE_DATATRANSFER;OPAL_METH_DXGETDESTDELIMITEDFIELDS;OPAL_METH_DXGETDESTDELIMITER;OPAL_METH_DXGETSOURCECHARSET;OPAL_METH_DXGETSOURCEDELIMITEDFIELDS;OPAL_METH_DXGETSOURCEDELIMITER;OPAL_METH_DXGETSOURCEFIELDNAMESFROMFIRST;OPAL_METH_DXGETSOURCENAME;OPAL_METH_DXGETSOURCERANGE;OPAL_METH_DXGETSOURCETYPE;OPAL_METH_DXSETDESTDELIMITEDFIELDS;OPAL_METH_DXSETDESTDELIMITER;OPAL_METH_DXSETDESTSEPARATOR;',0,"Defaultoverview",)} Related Topics

getSourceSeparator example

The following example uses **getSourceSeparator** to display the separator used in a field on the specified form. This confirms that the separator is set correctly and allows you to specify a new separator if necessary.

```
method pushButton(var eventInfo Event)
   var
      dt      DataTransfer
   endVar
   dt.setSource("iesimpld.txt")


;The following lines check to see what Corel Paradox determined as the type of
;of the source file. If it is delimited, Corel Paradox determines the separator,
;delimiter and which fields are delimited.
   switch
      case dt.getSourceType() = DTASCIIVar :
         fldType = "Delimited"
         fldDelimiter = dt.getSourceDelimiter()
         if dt.getSourceDelimitedFields() = DTDelimAllFields then
            fldDelimitedFields = "All"
         else
            fldDelimitedFields = "Text"
         endIF
         fldSeparator = dt.getSourceSeparator()
      case dt.getSourceType() = DTASCIIFixed :
         fldType = "Fixed"
      otherwise :
         msgInfo("Hello","File missing or not text.")
   endSwitch
endMethod
```

getSourceType method

Retrieves the source file type constant.

Syntax

```
getSourceType ( ) SmallInt
```

Description

getSourceType retrieves the source file type constant. The file type constant is one of the DataTransferFileType constants. The version part of the file type is irrelevant to the source and can be ignored.

Example

{button ,AL(`OPAL_TYPE_DATATRANSFER;OPAL_METH_DXGETDESTNAME;OPAL_METH_DXGETDESTTYPE;OPAL_METH_DXGETSOURCECHARSET;OPAL_METH_DXGETSOURCEDELIMITEDFIELDS;OPAL_METH_DXGETSOURCEDELIMITER;OPAL_METH_DXGETSOURCEFIELDNAMESFROMFIRST;OPAL_METH_DXGETSOURCENAME;OPAL_METH_DXGETSOURCERANGE;OPAL_METH_DXGETSOURCESEPARATOR;OPAL_METH_DXSETDEST;OPAL_METH_DXSETSOURCE;',0,"Defaultoverview",)} Related Topics

getSourceType example

The following example uses **getSourceType** to determine the file type of the source file:

```
method pushButton(var eventInfo Event)
   var
      dt      DataTransfer
   endVar
   dt.setSource("iesimpld.txt")


;The following lines check to see what Corel Paradox determined as the type of
;of the source file. If it is delimited, Corel Paradox determines the separator,
;delimiter and which fields are delimited.
   switch
      case dt.getSourceType() = DTASCIIVar :
         fldType = "Delimited"
         fldDelimiter = dt.getSourceDelimiter()
         if dt.getSourceDelimitedFields() = DTDelimAllFields then
            fldDelimitedFields = "All"
         else
            fldDelimitedFields = "Text"
         endIF
         fldSeparator = dt.getSourceSeparator()
      case dt.getSourceType() = DTASCIIFixed :
         fldType = "Fixed"
      otherwise :
         msgInfo("Hello","File missing or not text.")
   endSwitch
endMethod
```

importASCIIFix procedure

Imports data from a fixed record length ASCII text file to a table.

Syntax

```
importASCIIFix ( const fileName String, const tableName String, const specTableName String [ , const ANSI Logical ] ) Logical
```

Description

importASCIIFix imports data from an ASCII file in which each record's fields are the same length to a table. If the target table exists, its contents are replaced with the imported data. If the table does not exist, this method creates it. **importASCIIFix** duplicates the function of the Import Data dialog box.

The argument *fileName* specifies the source file and *tableName* specifies the target table. Dates and numbers are formatted according to your settings in the Windows Control Panel. The file extension specified in *tableName* identifies the table type of the target table. .DB specifies a Corel Paradox table and .DBF specifies a dBASE table. If you omit the extension, the data is imported to a Corel Paradox table by default.

The argument *specTableName* is the name of a table that specifies the layout for the imported data. The following table illustrates the structure specified in *specTableName* :

Field name	Type & size	Description
Field Name	A 25	Name of the field to import
Type	A 4	Field type to import. The Type must be a valid Corel Paradox or dBASE field specification.
Start	S	Column number where the field value begins
Length	S	Field size

ANSI (optional) specifies whether to use the ANSI or OEM character set. Set ANSI to True to specify the ANSI character set, and to False to specify the OEM (default) character set.

This method is part of the Data Transfer type, but in previous versions it was included in the System type.

Example

{button ,AL(`OPAL_TYPE_DATATRANSFER;OPAL_METH_DXAPPENDASCIIFIX;OPAL_METH_DXDLGIMPORTASCIIFIX;OPAL_METH_DXEXPORTASCIIFIX;OPAL_METH_DXIMPORTASCIIVAR;OPAL_METH_DXIMPORTSPREADSHEET;',0,"Defaultoverview",)} Related Topics

importASCIIFix example

The following example imports data from the ORDERS.TXT text file to the ORDERS.DB table. The ORDERS.DB table structure is read from the ORDERIMP.DB table and the OEM character set is used.

```
method pushButton ( var eventInfo Event )
   importASCIIFix ( "orders.txt", "orders.db", "orderimp.db", False )
endMethod
```

importASCIIVar procedure

Imports data from an ASCII text file with variable field length values to a table.

Syntax

```
importASCIIVar ( const fileName String, const tableName String [ , const separator String,
const delimiter String, const allFieldsDelimited Logical, const ANSI Logical ] ) Logical
```

Description

importASCIIVar imports data from an ASCII file to a table. The source file's variable length field values in each record may be delimited by an optionally specified character. If the target table exists, its contents are replaced with the imported data. If the table does not exist, this method creates it. **importASCIIVar** duplicates the function of the Import Data dialog box.

The argument *fileName* specifies the source file and *tableName* specifies the target table. Dates and numbers are formatted according to your settings in the Windows Control Panel. The file extension specified in *tableName* identifies the table type of the target table. .DB specifies a Corel Paradox table and .DBF specifies a dBASE table. If you omit the extension, the data is imported to a Corel Paradox table by default.

delimiter (optional) specifies the character that defines the limits of field values in the target. Leave the delimiter string empty if you do not want to define limits. The *allFieldsDelimited* (optional) string is marked True if data from all field types is delimited, and False if data from only text, alphanumeric, or character field types is delimited. Corel Paradox truncates strings longer than 255 characters when it imports them.

ANSI (optional) specifies whether to use the ANSI or OEM character set. Set *ANSI* to True to use the ANSI character set, or to False to use the OEM character set.

The following table displays the default settings for optional arguments:

separator	, (comma)
delimiter	" (double quote) text fields only
ANSI	False

This method is part of the Data Transfer type, but in previous versions it was included in the System type.

Example

{button ,AL(`OPAL_TYPE_DATATRANSFER;OPAL_METH_DXIMPORTASCIIFIX;OPAL_METH_DXIMPORTSPREADSHEET;',0,"Defaultoverview",)} Related Topics

importASCIIVar example

The following example imports data from the ORDERS.TXT text file to the ORDERS.DB table. In this example, commas delimit field values, values are not enclosed, and the ANSI character set is used.

```
method pushButton ( var eventInfo Event )
   importASCIIVar ( "orders.txt", "orders.db", ",", "", True, True )
endMethod
```

The following example imports ASCII delimited text to Corel Paradox (long form):

```
method pushButton ( var eventInfo Event )

var
   dt DataTransfer
endVar


dt.setSource("orders.txt", DTAsciiVar)
dt.setDest("orders.db")
dt.setSourceDelimiter("")
dt.setSourceSeparator(",")
dt.setSourceCharSet(dtANSI)
dt.setSourceDelimitedFields(dtDelimAllFields)


dt.TransferData()
endMethod
```

The following example imports ASCII delimited text to Corel Paradox (medium form):

```
method pushButton ( var eventInfo Event )

var
   dt DataTransfer
endVar


dt.setSource("NewRecords.txt", DTAsciiVar)
dt.setDest("TimeCards.db")
dt.TransferData()

endMethod
```

The following example imports ASCII Delimited Text to Corel Paradox (short form):

```
method pushButton ( var eventInfo Event )

ImportASCIIVar("NewRecords.txt", "TimeCards.db")

endMethod
```

importSpreadsheet procedure

Imports data from a spreadsheet file to a table.

Syntax

```
importSpreadsheet ( const fileName String, const tableName String, const fromCell String, const
toCell String [ , const getFieldNames Logical ] ) Logical
```

Description

importSpreadsheet imports the data from a spreadsheet file to a table. Corel Paradox converts rows to records and columns to fields. If the table does not exist prior to importing, this method creates it. **importSpreadsheet** duplicates the function of the Import Data dialog box.

fileName specifies the spreadsheet or source file, and *tableName* specifies the table that displays the imported data. *fromCell* specifies the upper-left cell and *toCell* specifies the lower-right cell of the imported block. *getFieldNames* specifies whether to format the top row of the spreadsheet as column headers for the table. If you set *getFieldNames* to True Corel Paradox creates column headers (default); If you set *getFieldNames* to False, Corel Paradox does not.

The file extension specified in *fileName* identifies the format of the spreadsheet file. The following table illustrates the file extensions and their spreadsheet formats:

Extension	Format
WB1, WB2, WB3	Quattro Pro Win
WQ1	Quattro Pro DOS
WKQ	Quattro
WK1	Lotus 2.x
WKS	Lotus 1.A
XLS	Excel 3.0/4.0/5.0

The file extension specified in *tableName* identifies the target table's type. .DB specifies a Corel Paradox table (default) and .DBF specifies a dBASE table.

Corel Paradox automatically assigns a field type to each column of data. The following table shows how Corel Paradox determines a field's type:

Spreadsheet value	Corel Paradox field type	dBASE field type
Label	Alpha	Character
Integer	Short	Float number (5,0)
Number	Number	Float number (20,4)
Currency	Money	Float number (20,4)
Date	Date	Date

The following rules determine which category a column falls into:

- A column containing a label (text) is converted to an alpha field (or character field for a dBASE table).
- A column containing both dates and numbers is converted to an alpha field (or character field for a dBASE table).
- A column containing only values formatted as currency is converted to a money field in a Corel Paradox table.
- A column containing both currency and number (or integer) values is converted to a number field.

Corel Paradox often imports dates and numbers from unedited spreadsheets as alpha fields. For example, spreadsheets often have rows of hyphens separating sections of numbers. Since only an alphanumeric field can have both numbers and hyphens, Corel Paradox converts each spreadsheet column to an alpha field even though it contains mostly numbers.

To avoid conversion problems, edit the spreadsheet before importing it. Follow these steps:

1. Remove extraneous entries such as hyphens, asterisks, and exclamation points.
2. Ensure each column contains one type of data and uses one formatting option.
3. Place the titles you want to format as table column headings in the top row of the selected range. Corel Paradox uses the first row that contains text to generate field names. If the spreadsheet does not contain column titles, set the *getFieldNames* parameter to False.

If the table does not have the format you want after you import it, restructure it in Corel Paradox.

This method is part of the Data Transfer type, but in previous versions it was included in the System type.

Example

{button ,AL(`OPAL_TYPE_DATATRANSFER;OPAL_METH_DXIMPORTASCIIFIX;OPAL_METH_DXIMPORTASCIIVAR;',0,"Defaultoverview",)} Related Topics

importSpreadsheet example

The following example imports data from a Quattro Pro for Windows file to the ORDERS.DB table. This example converts the first row of the spreadsheet file to column headers in the table.

```
method pushButton ( var eventInfo Event )
   importSpreadsheet ( "orders.wb1", "orders.db","A:A1", "A:H25", True )
endMethod
```

loadDestSpec method

Loads a fixed-length import file specification.

Syntax

```
loadDestSpec ( const tableName String )
```

Description

loadDestSpec loads a fixed-length import file specification. The argument *tableName* specifies the table to use as the pattern for the destination specification. **loadDestSpec** applies only when the destination is a fixed-length ASCII text file.

Example

{button ,AL(`OPAL_TYPE_DATATRANSFER;OPAL_METH_DXLOADSOURCESPEC;OPAL_METH_DXTRANSFERDATA;',0,"Defaultoverview",)} Related Topics

loadDestSpec example

The following examples specify a DataTransfer data type. Use this code to build an Import or Export specification and then call the **transferData** method.

Import from text

```
var
   dt DataTransfer
endVar
dt.SetSource ( "MYFILE.TXT" )
if dt.getSourceType ( ) = DTASCIIFixed Then
   dt.loadDestSpec ( "SpecTable" )
EndIf
dt.setDest ( "Existing Data.db" )
dt.setAppend ( True )
dt.transferData ( )
```

loadSourceSpec method

Loads a fixed-length import file specification.

Syntax

```
loadSourceSpec ( const tableName String )
```

Description

loadSourceSpec loads a fixed-length import file specification. The argument *tableName* specifies the table to use as the pattern for the source specification. **loadSourceSpec** applies only when the source is a fixed-length ASCII text file.

Example

{button ,AL(`OPAL_TYPE_DATATRANSFER;OPAL_METH_DXLOADDESTSPEC;OPAL_METH_DXTRANSFERDATA;',0,"Defaultoverview",)} Related Topics

loadSourceSpec example

The following examples specify a DataTransfer data type. Use this code to build an Import or Export specification, and then call the **transferData** method.

Import from text

```
var
   dt DataTransfer
endVar
dt.SetSource ( "MYFILE.TXT" )
if dt.getSourceType ( ) = DTASCIIFixed Then
   dt.loadSourceSpec ( "SpecTable" )
EndIf
dt.setSource ( "Existing Data.db" )
dt.setAppend ( True )
dt.transferData ( )
```

setAppend method

Appends data to the existing table.

Syntax

```
setAppend ( const AppendToTable Logical )
```

Description

setAppend appends data to the existing table when set to True, and overwrites the table when set to False. This method is ignored for new tables, and applies only when the destination is a table.

Example

{button ,AL(`OPAL_TYPE_DATATRANSFER;OPAL_METH_DXAPPENDASCIIFIX;OPAL_METH_DXAPPENDASCIIVAR;OPAL_METH_DXDLGIMPORT;OPAL_METH_DXDLGIMPORTASCIIFIX;OPAL_METH_DXDLGIMPORTASCIIVAR;OPAL_METH_DXDLGIMPORTSPREADSHEET;OPAL_METH_DXDLGIMPORTTABLE;OPAL_METH_DXGETAPPEND;OPAL_METH_DXIMPORTASCIIFIX;OPAL_METH_DXIMPORTASCIIVAR;OPAL_METH_DXIMPORTSPREADSHEET;',0,"Defaultoverview",)} Related Topics

setAppend example

The following example specifies a DataTransfer data type. Use this code to build an Import or Export specification and then call the **transferData** method.

Import from text

```
var
   dt DataTransfer
endVar
dt.SetSource ( "MYFILE.TXT" )
if dt.getSourceType ( ) = DTASCIIFixed Then
   dt.loadDestSpec ( "SpecTable" )
EndIf
dt.setDest ( "Existing Data.db" )
dt.setAppend ( True )
dt.transferData ( )
```

setDest method

Specifies the file or table to receive imported data.

Syntax

```
setDest ( const destName String, [ const destType SmallInt ] )
```

Description

setDest specifies the name and type of the file or table that receives data. If no file type is specified, the file extension determines its type. The file type constant *destType* specifies one of the DataTransferFileType constants.

Example

{button ,AL(`OPAL_TYPE_DATATRANSFER;OPAL_METH_DXGETDESTNAME;OPAL_METH_DXGETDESTTYPE;OPAL_METH_DXGETSOURCENAME;OPAL_METH_DXGETSOURCETYPE;OPAL_METH_DXSETDESTCHARSET;OPAL_METH_DXSETDESTDELIMITEDFIELDS;OPAL_METH_DXSETDESTDELIMITER;OPAL_METH_DXSETDESTFIELDNAMESFROMFIRST;OPAL_METH_DXSETDESTSEPARATOR;OPAL_METH_DXSETSOURCE;',0,"Defaultoverview",)} Related Topics

setDest example

The following example specifies a DataTransfer data type. Use this code to build an Import or Export specification, and then call the **transferData** method.

Import from spreadsheet

```
var
   dt DataTransfer
endVar
dt.setSource ( "MYFILE.WKS" )
dt.setDest ( "New Data.db" )
dt.setProblems ( True )
dt.transferData ( )
```

setDestCharSet method

Sets the file character set to dtOEM or dtANSI.

Syntax

```
setDestCharSet ( const CharSetCode SmallInt )
```

Description

setDestCharSet sets the file character set to one of the two DataTransferCharset constants: dtOEM or dtANSI. This method applies only when the destination is a fixed or delimited ASCII text file.

Example

{button ,AL(`OPAL_TYPE_DATATRANSFER;OPAL_METH_DXENUMSOURCERANGELIST;OPAL_METH_DXGETDESTCHARSET;OPAL_METH_DXGETSOURCERANGE;OPAL_METH_DXSETDEST;OPAL_METH_DXSETDESTDELIMITEDFIELDS;OPAL_METH_DXSETDESTDELIMITER;OPAL_METH_DXSETDESTFIELDNAMESFROMFIRST;OPAL_METH_DXSETDESTSEPARATOR;OPAL_METH_DXSETSOURCERANGE;',0,"Defaultoverview",)} Related Topics

setDestCharSet example

The following example uses the **transferData** method to export ORDERS.DB to ORDINFO.TXT. This example uses **setDestCharSet** to set the file character set to ANSI. To set the file character set to OEM, use the DTOEM constant with **setDestCharSet**.

```
method pushButton(var eventInfo Event)
   var
      dt      DataTransfer
   endVar
   dt.setDest("ordinfo.txt", DTASCIIVar)
   dt.setSource("orders.db")


   ;Specify the single quote (') to surround the fields.
   ;The delimited fields will be text fields only.
   dt.setDestDelimiter("'")
   dt.setDestDelimitedFields(DTDelimJustText)


   ;Specify the tab character to separate the fields.
   dt.setDestSeparator("\t")


   ;Set the first row of the ORDINFO.TXT to be the field names
   dt.setDestFieldNamesFromFirst(True)


   ;Set the character set of the destination file ORDINFO.TXT to be the ANSI
   ;character set.
   dt.setDestCharSet(DTAnsi)


   ;run Export
   dt.transferData()
endMethod
```

setDestDelimitedFields method

Sets the Delimited Fields setting to DtDelimAllFields or DtDelimJustText.

Syntax

```
setDestDelimitedFields ( const delimitCode SmallInt )
```

Description

setDestDelimitedFields sets the Delimited Fields setting. The argument *delimitCode* specifies one of the two DataTransferDelimitCode constants: DtDelimAllFields or DtDelimJustText. **setDestDelimitedFields** only applies when the destination is a delimited ASCII text file.

Example

{button ,AL(`OPAL_TYPE_DATATRANSFER;OPAL_METH_DXGETDESTDELIMITEDFIELDS;OPAL_METH_DXGETDESTDELIMITER;OPAL_METH_DXGETDESTSEPARATOR;OPAL_METH_DXSETDEST;OPAL_METH_DXSETDESTCHARSET;OPAL_METH_DXSETDESTDELIMITER;OPAL_METH_DXSETDESTFIELDNAMESFROMFIRST;OPAL_METH_DXSETDESTSEPARATOR;',0,"Defaultoverview",)} Related Topics

setDestDelimitedFields example

The following example uses the **transferData** method to export ORDERS.DB to ORDINFO.TXT. It uses **setDestDelimitedFields** to delimit surrounding text fields only. To delimit all fields, use the DTDelimAllFields constant with **setDestDelimitedFields**.

```
method pushButton(var eventInfo Event)
   var
      dt      DataTransfer
   endVar
   dt.setDest("ordinfo.txt", DTASCIIVar)
   dt.setSource("orders.db")


   ;Specify the single quote (') to surround the fields.
   ;The delimited fields will be text fields only.
   dt.setDestDelimiter("'")
   dt.setDestDelimitedFields(DTDelimJustText)


   ;Specify the tab character to separate the fields.
   dt.setDestSeparator("\t")


   ;Set the first row of the ORDINFO.TXT to be the field names
   dt.setDestFieldNamesFromFirst(True)


   ;Set the character set of the destination file ORDINFO.TXT to be the ANSI
   ;character set.
   dt.setDestCharSet(DTAnsi)


   ;run Export
   dt.transferData()
endMethod
```

setDestDelimiter method

Specifies a character as the delimiter.

Syntax

```
setDestDelimiter ( const delimiterChar String )
```

Description

setDestDelimiter sets the delimiter to the character specified by *delimiterChar*. The default delimiter is a comma. **setDestDelimiter** only applies when the destination is a delimited ASCII text file.

Example

{button ,AL(`OPAL_TYPE_DATATRANSFER;OPAL_METH_DXGETDESTDELIMITEDFIELDS;OPAL_METH_DXGETDESTDELIMITER;OPAL_METH_DXGETDESTSEPARATOR;OPAL_METH_DXSETDEST;OPAL_METH_DXSETDESTCHARSET;OPAL_METH_DXSETDESTDELIMITEDFIELDS;OPAL_METH_DXSETDESTFIELDNAMESFROMFIRST;OPAL_METH_DXSETDESTSEPARATOR;',0,"Defaultoverview",)} Related Topics

setDestDelimiter example

The following example exports the ORDERS.DB table into an ASCII delimited text file. The delimiter single quote character is the specified delimiter.

```
method pushButton(var eventInfo Event)
   var
      dt      DataTransfer
   endVar
   dt.setDest("ordinfo.txt", DTASCIIVar)
   dt.setSource("orders.db")


   ;Specify the single quote (') to surround the fields.
   ;The delimited fields will be text fields only.
   dt.setDestDelimiter("'")
   dt.setDestDelimitedFields(DTDelimJustText)


   ;Specify the tab character to separate the fields.
   dt.setDestSeparator("\t")


   ;Set the first row of the ORDINFO.TXT to be the field names
   dt.setDestFieldNamesFromFirst(True)


   ;Set the character set of the destination file ORDINFO.TXT to be the ANSI
   ;character set.
   dt.setDestCharSet(DTAnsi)


   ;run Export
   dt.transferData()
endMethod
```

setDestFieldNamesFromFirst method

Sets field names using the data in the first row of input.

Syntax

```
setDestFieldNamesFromFirst ( const namesFirst Logical )
```

Description

setDestFieldNamesFromFirst sets the first row of the destination file to be the field names of the table. Setting *namesFirst* to True creates the first row as field names and data will begin on the second row.

setDestFieldNamesFromFirst applies to both spreadsheets and delimited text files.

Example

{button ,AL(`OPAL_TYPE_DATATRANSFER;OPAL_METH_DXDLGIMPORTSPREADSHEET;OPAL_METH_DXEXPORTSPREADSHEET;OPAL_METH_DXGETDESTFIELDNAMESFROMFIRST;OPAL_METH_DXIMPORTSPREADSHEET;OPAL_METH_DXSETDEST;OPAL_METH_DXSETDESTCHARSET;OPAL_METH_DXSETDESTDELIMITEDFIELDS;OPAL_METH_DXSETDESTDELIMITER;OPAL_METH_DXSETDESTSEPARATOR;',0,"Defaultoverview",)}

Related Topics

setDestFieldNamesFromFirst example

The following example uses the **transferData** method to export ORDERS.DB to ORDINFO.TXT. The **setDestFieldNamesFromFirst** is used to set the field names using the data in the first row of the text file.

```
method pushButton(var eventInfo Event)
   var
      dt      DataTransfer
   endVar
   dt.setDest("ordinfo.txt", DTASCIIVar)
   dt.setSource("orders.db")


   ;Specify the single quote (') to surround the fields.
   ;The delimited fields will be text fields only.
   dt.setDestDelimiter("'")
   dt.setDestDelimitedFields(DTDelimJustText)


   ;Specify the tab character to separate the fields.
   dt.setDestSeparator("\t")


   ;Set the first row of the ORDINFO.TXT to be the field names
   dt.setDestFieldNamesFromFirst(True)


   ;Set the character set of the destination file ORDINFO.TXT to be the ANSI
   ;character set.
   dt.setDestCharSet(DTAnsi)


   ;run Export
   dt.transferData()
endMethod
```

setDestSeparator method

Sets the separator character for delimited ASCII text.

Syntax

```
setDestSeparator ( const separatorChar String )
```

Description

setDestSeparator sets the separator to the character specified by *separatorChar*. The default separator is the comma character. **setDestSeparator** only applies when the destination is a delimited ASCII text file.

Example

{button ,AL(`OPAL_TYPE_DATATRANSFER;OPAL_METH_DXGETDESTDELIMITEDFIELDS;OPAL_METH_DXGETDESTDELIMITER;OPAL_METH_DXGETDESTSEPARATOR;OPAL_METH_DXSETDEST;OPAL_METH_DXSETDESTCHARSET;OPAL_METH_DXSETDESTDELIMITEDFIELDS;OPAL_METH_DXSETDESTDELIMITER;OPAL_METH_DXSETDESTFIELDNAMESFROMFIRST;',0,"Defaultoverview",)} Related Topics

setDestSeparator example

The following example specifies a DataTransfer data type. Use this code to build an Import or Export specification, and then call the **transferData** method.

Export to text

```
var
   dt DataTransfer
endVar
msgInfo("Info", "The current source separator is " + dt.getSourceSeparator())
dt.setSource ( "ANSWER.DB" )
msgInfo("Info", "The current destination separator is " + dt.getDestSeparator())
dt.SetDest ( "NEWFILE.TXT",dtASCIIVar )
dt.setDestSeparator ( ";" )
dt.transferData ( )
```

setKeyviol method

Writes violations to the Keyviol table.

Syntax

```
setKeyviol ( const GenerateKeyviol Logical )
```

Description

setKeyviol writes violations to the Keyviol table. The argument *generateKeyviol* is a logical that is set to True to write violations to the Keyviol table. *generateKeyviol* is ignored for tables without keys. **setKeyviol** applies only when the destination is a table.

Example

{button ,AL(`OPAL_TYPE_DATATRANSFER;OPAL_METH_DXSETPROBLEMS;OPAL_METH_DXGETKEYVIOL;OPAL_METH_DXGETPROBLEMS;',0,"Defaultoverview",)} Related Topics

setKeyviol example

The following example specifies a DataTransfer data type. Use this code to build an Import or Export specification and then call the **transferData** method.

Imports ASCII delimited text to Corel Paradox (long form):

```
var
   dt DataTransfer
endVar


; Fields Quoted even if numeric
dt.setAppend(True)      ; Append to an existing Table
dt.setProblems(True)    ; Generate a Problems Table (if Any)
dt.setKeyviol(True)     ; Generate a Keyviol Table (if any)


dt.setSource("NewRecords.txt", DTAsciiVar)
dt.setDest("TimeCards.db")
dt.TransferData()
```

setProblems method

Writes problems to a specified table.

Syntax

```
setProblems ( const generateProblems Logical )
```

Description

setProblems writes problems to the Problems table. This method applies only when the destination is a table.

Example

{button ,AL(`OPAL_TYPE_DATATRANSFER;OPAL_METH_DXSETKEYVIOL;OPAL_METH_DXGETKEYVIOL;OPAL_METH_DXGETPROBLEMS;',0,"Defaultoverview",)} Related Topics

setProblems example

The following example specifies a DataTransfer data type. Use this code to build an Import or Export specification and then call the **transferData** method.

Imports ASCII Delimited Text to Corel Paradox (long form):

```
var
   dt DataTransfer
endVar


; Fields Quoted even if numeric
dt.setAppend(True)      ; Append to an existing Table
dt.setProblems(True)    ; Generate a Problems Table (if Any)
dt.setKeyviol(True)     ; Generate a Keyviol Table (if any)


dt.setSource("NewRecords.txt", DTAsciiVar)
dt.setDest("TimeCards.DB")
dt.TransferData()
```

setSource method

Specifies the file or table that acts as the data source and its type.

Syntax

```
setSource ( const sourceName String, [ const sourceType SmallInt ] )
```

Description

setSource specifies the file or table to use as the data source. *SourceType* specifies the field type according to the application that generated the file. If no file type is specified, the *sourceName* file extension is used to determine the file's type. The file type constant *destType* specifies one of the DataTransferFileType constants.

Example

{button ,AL(`OPAL_TYPE_DATATRANSFER;OPAL_METH_DXGETDESTNAME;OPAL_METH_DXGETDESTTYPE;OPAL_METH_DXGETSOURCENAME;OPAL_METH_DXGETSOURCETYPE;OPAL_METH_DXSETSOURCECHARSET;OPAL_METH_DXSETSOURCEDELIMITEDFIELDS;OPAL_METH_DXSETSOURCEDELIMITER;OPAL_METH_DXSETSOURCEFIELDNAMESFROMFIRST;OPAL_METH_DXSETSOURCERANGE;OPAL_METH_DXSETSOURCESEPARATOR;',0,"Defaultoverview",)} Related Topics

setSource example

The following example specifies a DataTransfer data type. Use this code to build an Import or Export specification and then call the **transferData** method.

Import from spreadsheet

```
var
   dt DataTransfer
endVar
dt.setSource ( "MYFILE.WKS" )
dt.setDest ( "New Data.db" )
dt.setProblems ( True )
dt.transferData ( )
```

setSourceCharSet method

Sets the file character set to dtOEM or dtANSI.

Syntax

```
setSourceCharSet ( const charSetCode SmallInt )
```

Description

setSourceCharSet sets the file character set. The argument *charSetCode* specifies one of the two DataTransferCharset constants: dtOEM or dtANSI. **setSourceCharSet** applies only when the source is a fixed or delimited ASCII text file.

Example

{button ,AL(`OPAL_TYPE_DATATRANSFER;OPAL_METH_DXENUMSOURCERANGELIST;OPAL_METH_DXGETSOURCECHARSET;OPAL_METH_DXGETSOURCERANGE;OPAL_METH_DXSETSOURCE;OPAL_METH_DXSETSOURCEDELIMITEDFIELDS;OPAL_METH_DXSETSOURCEDELIMITER;OPAL_METH_DXSETSOURCEFIELDNAMESFROMFIRST;OPAL_METH_DXSETSOURCERANGE;OPAL_METH_DXSETSOURCESEPARATOR;',0,"Defaultoverview",)} Related Topics

setSourceCharSet example

The following example specifies a DataTransfer data type. This structure is used with the transferData method. This example assumes that the DataTransfer variable, dt, is declared within a Var ... EndVar statement. The custom method cmTransfer() is within the scope of the variable (dt).

```
method cmTransfer()   ;this example completes a DataTransfer


   dt.setSource("CUSTOMER.TXT", DTASCIIVar)  ; sets the datatransfer source
                                             ; to CUSTOMER.TXT
   dt.setSourceSeparator("/")   ; specifies the forward slash "/" character
                                ; to separate each field
   dt.setSourceDelimiter("'")   ; specifies the single quote to surround
                                ; the fields
   dt.setSourceDelimitedFields(DTDelimJustText)  ; specifies that the single
                                                 ; quote (delimiter) surrounds
                                                 ; only text fields of the
                                                 ; source file
   dt.setSourceCharSet(DTANSI)   ; specifies that the character set used
                                 ; when creating the source file
                                 ; was the ANSI character set


   dt.setSourceFieldNamesFromFirst(False)   ; specifies to use the first
                                            ; row of the source file as
                                            ; field names
   dt.setDest("NEWCUST.DB")   ; sets the destination file to NEWCUST.DB
   dt.setProblems(True)   ; specifies to create a PROBLEMS.DB if there are
                          ; any problems importing the source file
   dt.transferData()   ; executes the data transfer. In this case it
                       ; imports the CUSTOMER.TXT file as NEWCUST.DB.
   dt.empty()   ; empties the dt variable structure to set it up for
                ; a new transfer.

endmethod
```

setSourceDelimitedFields method

Sets the Delimited Fields setting to DtDelimAllFields or DtDelimJustText.

Syntax

```
setSourceDelimitedFields ( const delimitCode SmallInt )
```

Description

setSourceDelimitedFields sets the delimited fields value. The argument *delimitCode* specifies one of the two DataTransferDelimitCode constants: DtDelimAllFields or DtDelimJustText. **setSourceDelimitedFields** only applies when the source or the destination is a delimited ASCII text file.

Example

{button ,AL(`OPAL_TYPE_DATATRANSFER;OPAL_METH_DXGETSOURCEDELIMITEDFIELDS;OPAL_METH_DXGETSOURCEDELIMITER;OPAL_METH_DXGETSOURCESEPARATOR;OPAL_METH_DXSETSOURCE;OPAL_METH_DXSETSOURCECHARSET;OPAL_METH_DXSETSOURCEDELIMITER;OPAL_METH_DXSETSOURCEFIELDNAMESFROMFIRST;OPAL_METH_DXSETSOURCERANGE;OPAL_METH_DXSETSOURCESEPARATOR;',0,"Defaultoverview",)} Related Topics

setSourceDelimitedFields example

The following example specifies a DataTransfer data type. This structure is used with the **transferData** method. This example assumes that the DataTransfer variable, dt, is declared within a Var ... EndVar statement. The custom method **cmTransfer()** is within the scope of the variable (dt).

```
method cmTransfer()   ;this example completes a DataTransfer


   dt.setSource("CUSTOMER.TXT", DTASCIIVar)  ; sets the datatransfer source
                                             ; to CUSTOMER.TXT
   dt.setSourceSeparator("/")   ; specifies the forward slash "/" character
                                ; to separate each field
   dt.setSourceDelimiter("'")   ; specifies the single quote to surround
                                ; the fields
   dt.setSourceDelimitedFields(DTDelimJustText)  ; specifies that the single
                                                 ; quote (delimiter) surrounds
                                                 ; only text fields of the
                                                 ; source file
   dt.setSourceCharSet(DTANSI)   ; specifies that the character set used
                                 ; when creating the source file
                                 ; was the ANSI character set


   dt.setSourceFieldNamesFromFirst(False)   ; specifies to use the first
                                            ; row of the source file as
                                            ; field names
   dt.setDest("NEWCUST.DB")   ; sets the destination file to NEWCUST.DB
   dt.setProblems(True)   ; specifies to create a PROBLEMS.DB if there are
                          ; any problems importing the source file
   dt.transferData()   ; executes the data transfer. In this case it
                       ; imports the CUSTOMER.TXT file as NEWCUST.DB.
   dt.empty()   ; empties the dt variable structure to set it up for
                ; a new transfer.

endmethod
```

setSourceDelimiter method

Specifies a character as the delimiter.

Syntax

```
setSourceDelimiter ( const delimiterChar String )
```

Description

setSourceDelimiter sets the delimiter to the character specified by *delimiterChar*. The default delimiter is a comma. **setSourceDelimiter** only applies when the source or the destination is a delimited ASCII text file.

Example

{button ,AL(`OPAL_TYPE_DATATRANSFER;OPAL_METH_DXGETSOURCEDELIMITEDFIELDS;OPAL_METH_DXGETSOURCEDELIMITER;OPAL_METH_DXGETSOURCESEPARATOR;OPAL_METH_DXSETSOURCE;OPAL_METH_DXSETSOURCECHARSET;OPAL_METH_DXSETSOURCEDELIMITEDFIELDS;OPAL_METH_DXSETSOURCEFIELDNAMESFROMFIRST;OPAL_METH_DXSETSOURCERANGE;OPAL_METH_DXSETSOURCESEPARATOR;',0,"Defaultoverview",)} Related Topics

setSourceDelimiter example

The following example specifies a DataTransfer data type. This structure is used with the **transferData** method. This example assumes that the DataTransfer variable, dt, is declared within a Var ... EndVar statement. The custom method **cmTransfer()** is within the scope of the variable (dt).

```
method cmTransfer()   ;this example completes a DataTransfer


   dt.setSource("CUSTOMER.TXT", DTASCIIVar)  ; sets the datatransfer source
                                             ; to CUSTOMER.TXT
   dt.setSourceSeparator("/")   ; specifies the forward slash "/" character
                                ; to separate each field
   dt.setSourceDelimiter("'")   ; specifies the single quote to surround
                                ; the fields
   dt.setSourceDelimitedFields(DTDelimJustText)  ; specifies that the single
                                                 ; quote (delimiter) surrounds
                                                 ; only text fields of the
                                                 ; source file
   dt.setSourceCharSet(DTANSI)   ; specifies that the character set used
                                 ; when creating the source file
                                 ; was the ANSI character set


   dt.setSourceFieldNamesFromFirst(False)   ; specifies to use the first
                                            ; row of the source file as
                                            ; field names
   dt.setDest("NEWCUST.DB")   ; sets the destination file to NEWCUST.DB
   dt.setProblems(True)   ; specifies to create a PROBLEMS.DB if there are
                          ; any problems importing the source file
   dt.transferData()   ; executes the data transfer. In this case it
                       ; imports the CUSTOMER.TXT file as NEWCUST.DB.
   dt.empty()   ; empties the dt variable structure to set it up for
                ; a new transfer.

endmethod
```

setSourceFieldNamesFromFirst method

Sets field names using the data in the first row of input.

Syntax

```
setSourceFieldNamesFromFirst ( const namesFirst Logical)
```

Description

setSourceFieldNamesFromFirst sets field names using the first row of the input data. Setting *namesFirst* to True always skips the first row. However, the field names only apply to newly created tables that do not already have field names. **setSourceFieldNamesFromFirst** only applies when the source is a spreadsheet.

Example

{button ,AL(`OPAL_TYPE_DATATRANSFER;OPAL_METH_DXDLGIMPORTSPREADSHEET;OPAL_METH_DXEXPORTSPREADSHEET;OPAL_METH_DXGETSOURCEFIELDNAMESFROMFIRST;OPAL_METH_DXIMPORTSPREADSHEET;OPAL_METH_DXSETSOURCE;OPAL_METH_DXSETSOURCECHARSET;OPAL_METH_DXSETSOURCEDELIMITEDFIELDS;OPAL_METH_DXSETSOURCEDELIMITER;OPAL_METH_DXSETSOURCERANGE;OPAL_METH_DXSETSOURCESEPARATOR;',0,"Defaultoverview",)} Related Topics

setSourceFieldNamesFromFirst example

The following example specifies a DataTransfer data type. This structure is used with the **transferData** method. This example assumes that the DataTransfer variable, dt, is declared within a Var ... EndVar statement. The custom method **cmTransfer()** is within the scope of the variable (dt).

```
method cmTransfer()   ;this example completes a DataTransfer


   dt.setSource("CUSTOMER.TXT", DTASCIIVar)  ; sets the datatransfer source
                                             ; to CUSTOMER.TXT
   dt.setSourceSeparator("/")   ; specifies the forward slash "/" character
                                ; to separate each field
   dt.setSourceDelimiter("'")   ; specifies the single quote to surround
                                ; the fields
   dt.setSourceDelimitedFields(DTDelimJustText)  ; specifies that the single
                                                 ; quote (delimiter) surrounds
                                                 ; only text fields of the
                                                 ; source file
   dt.setSourceCharSet(DTANSI)   ; specifies that the character set used
                                 ; when creating the source file
                                 ; was the ANSI character set


   dt.setSourceFieldNamesFromFirst(False)   ; specifies to use the first
                                            ; row of the source file as
                                            ; field names
   dt.setDest("NEWCUST.DB")   ; sets the destination file to NEWCUST.DB
   dt.setProblems(True)   ; specifies to create a PROBLEMS.DB if there are
                          ; any problems importing the source file
   dt.transferData()   ; executes the data transfer. In this case it
                       ; imports the CUSTOMER.TXT file as NEWCUST.DB.
   dt.empty()   ; empties the dt variable structure to set it up for
                ; a new transfer.

endmethod
```

setSourceRange method

Specifies a sub range of the spreadsheet to import.

Syntax

```
setSourceRange ( const range String )
```

Description

setSourceRange specifies a sub range of the spreadsheet to import. The value specified by **setSourceRange** can be a named range, a page name, or an explicit range in QPW or Excel format. **setSourceRange** only applies when the source is a spreadsheet.

Example

{button ,AL(`OPAL_TYPE_DATATRANSFER;OPAL_METH_DXENUMSOURCERANGELIST;OPAL_METH_DXGETDESTCHARSET;OPAL_METH_DXGETSOURCERANGE;OPAL_METH_DXSETDESTCHARSET;OPAL_METH_DXSETSOURCE;OPAL_METH_DXSETSOURCECHARSET;OPAL_METH_DXSETSOURCEDELIMITEDFIELDS;OPAL_METH_DXSETSOURCEDELIMITER;OPAL_METH_DXSETSOURCEFIELDNAMESFROMFIRST;OPAL_METH_DXSETSOURCESEPARATOR;',0,"Defaultoverview",)} **Related Topics**

setSourceRange example

The following example uses the **setSourceRange** method to specify a range in a spreadsheet to import. You can specify named ranges and standard ranges.

```
method pushButton(var eventInfo Event)
   var
      dt      DataTransfer
   endVar
   dt.setSource("092595.wb2")


   ;Set the range to import from the spreadsheet.
   ;Either named range or specified range (ie. Page1:A1..Page3:AB10)
   dt.setSourceRange("myRange")
   dt.setSourceFieldNamesFromFirst(True)
   dt.setDest("delme09.db")


   ;Prompt the user to verify range to import. getSourceRange returns the
   ;actual range notation.
   view(dt.getSourceRange(),"Import Range")
   dt.transferData()
endMethod
```

setSourceSeparator method

Sets the separator character for delimited ASCII text.

Syntax

```
setSourceSeparator ( const separatorChar String )
```

Description

setSourceSeparator sets the separator to the character specified by *separatorChar*. The default separator is a comma. **setSourceSeparator** only applies when the source or the destination is a delimited ASCII text file.

Example

{button ,AL(`OPAL_TYPE_DATATRANSFER;OPAL_METH_DXGETSOURCEDELIMITEDFIELDS;OPAL_METH_DXGETSOURCEDELIMITER;OPAL_METH_DXGETSOURCESEPARATOR;OPAL_METH_DXSETSOURCE;OPAL_METH_DXSETSOURCECHARSET;OPAL_METH_DXSETSOURCEDELIMITEDFIELDS;OPAL_METH_DXSETSOURCEDELIMITER;OPAL_METH_DXSETSOURCEFIELDNAMESFROMFIRST;OPAL_METH_DXSETSOURCERANGE;',0,"Default overview",)} **Related Topics**

setSourceSeparator example

The following example specifies a DataTransfer data type. This structure is used with the **transferData** method. This example assumes that the DataTransfer variable, dt, is declared within a Var ... EndVar statement. The custom method **cmTransfer()** is within the scope of the variable (dt).

```
method cmTransfer()   ;this example completes a DataTransfer


   dt.setSource("CUSTOMER.TXT", DTASCIIVar)  ; sets the datatransfer source
                                             ; to CUSTOMER.TXT
   dt.setSourceSeparator("/")   ; specifies the forward slash "/" character
                                ; to separate each field
   dt.setSourceDelimiter("'")   ; specifies the single quote to surround
                                ; the fields
   dt.setSourceDelimitedFields(DTDelimJustText)  ; specifies that the single
                                                 ; quote (delimiter) surrounds
                                                 ; only text fields of the
                                                 ; source file
   dt.setSourceCharSet(DTANSI)   ; specifies that the character set used
                                 ; when creating the source file
                                 ; was the ANSI character set


   dt.setSourceFieldNamesFromFirst(False)   ; specifies to use the first
                                            ; row of the source file as
                                            ; field names
   dt.setDest("NEWCUST.DB")   ; sets the destination file to NEWCUST.DB
   dt.setProblems(True)   ; specifies to create a PROBLEMS.DB if there are
                          ; any problems importing the source file
   dt.transferData()   ; executes the data transfer. In this case it
                       ; imports the CUSTOMER.TXT file as NEWCUST.DB.
   dt.empty()   ; empties the dt variable structure to set it up for
                ; a new transfer.

endmethod
```

transferData method

Copies data from the source to the target.

Syntax

```
transferData ( )
```

Description

transferData copies data from the source to the destination. This method applies only if the source, the destination, or both the source and the destination are tables.

Example

{button ,AL(`OPAL_TYPE_DATATRANSFER;OPAL_METH_DXAPPENDASCIIFIX;OPAL_METH_DXAPPENDASCIIVAR;OPAL_METH_DXDLGEXPORT;OPAL_METH_DXDLGIMPORTASCIIFIX;OPAL_METH_DXDLGIMPORTASCIIVAR;OPAL_METH_DXDLGIMPORT;OPAL_METH_DXDLGIMPORTSPREADSHEET;OPAL_METH_DXDLGIMPORTTABLE;OPAL_METH_DXIMPORTASCIIFIX;OPAL_METH_DXIMPORTASCIIVAR;OPAL_METH_DXIMPORTSPREADSHEET;OPAL_METH_DXLOADDESTSPEC;',0,"Defaultoverview",)} Related Topics

transferData example

The following examples specify a DataTransfer data type. Use this code to build an Import or Export specification and then call the **transferData** method.

Import from spreadsheet

```
var
   dt DataTransfer
endVar
dt.setSource ( "MYFILE.WKS" )
dt.setDest ( "New Data.db" )
dt.setProblems ( True )
dt.transferData ( )
```

Import from text

```
var
   dt DataTransfer
endVar
dt.SetSource ( "MYFILE.TXT" )
if dt.getSourceType ( ) = DTASCIIFixed Then
   dt.loadDestSpec ( "SpecTable" )
EndIf
dt.setDest ( "Existing Data.db" )
dt.setAppend ( True )
dt.transferData ( )
```

Export to text

```
var
   dt DataTransfer
endVar
dt.setSource ( "ANSWER.DB" )
dt.SetDest ( "NEWFILE.TXT" )
dt.setDestSeparator ( ";" )
dt.transferData ( )
```

Toolbar type

The Toolbar type contains methods that create, delete, manipulate, and modify Toolbars.

The Toolbar type includes several derived methods from the AnyType type.

Methods for the Toolbar type

AnyType	**Toolbar**
blank	**addButton**
dataType	**attach**
isAssigned	**create**
isBlank	**createTabbed**
isFixedType	**empty**
unAssign	**getPosition**
	getState
	hide
	isAppBarVisible
	isVisible
	remove
	removeButton
	setPosition
	setState
	show
	showApplicationBar
	unAttach

Print related ObjectPAL methods and examples

addButton method

Adds a button to a Toolbar.

Syntax

```
1. addButton ( const idCluster SmallInt, const buttonType SmallInt, const idCommand SmallInt,
const grBmp Graphic, const buttonHelp String [ , const status String ] ) Logical

2. addButton ( const idCluster SmallInt, const buttonType SmallInt, const idCommand SmallInt,
const idBmp SmallInt, const buttonHelp String [ , const status String ] ) Logical
```

Description

addButton adds a button to a Toolbar. The new button's position is specified by a Cluster Identifier named *idCluster* . *idCluster* is an integer that ranges from 0 to 12. The type of button added is specified by *buttonType*. Button types include pushbutton, radio button, toggle button, and repeat button. When the button is pressed, the menu command that is sent is specified by *idCommand*. The contents of the small popup window that appears when the cursor is placed on the new button is specified by*buttonHelp*.

An optional parameter *status* (added in version 8) allows you to specify a different message to display on the status line. If *status* is omitted, the status line displays the text indicated for *buttonHelp*.

Syntax 1 adds a button to the Toolbar using a graphic bitmap (*grBmp*) to specify the button’s image on the Toolbar. This allows you to use a user-defined bitmap file or a bitmap object of a graphic type stored in a table.

Syntax 2 adds a button to the Toolbar using a bitmap constant. The bitmap constant specifies the button’s image on the Toolbar. This method allows you to create a button using any of the defined Toolbar button bitmaps in the system resource.

The only item that can be added to a Toolbar is a button.

addButton returns True if the button is successfully created.

Example

{button ,AL(`OPAL_TYPE_TOOLBAR;OPAL_METH_TOEMPTY;OPAL_METH_TOREMOVEBUTTON;',0,"Defaultoverview",)} Related Topics

addButton example

The following example creates a Toolbar named Edit and adds three buttons to the Toolbar using defined Corel Paradox bitmap constants:

```
method pushButton (var eventInfo Event)
var
   tb  Toolbar
endvar


   ;// Create a Toolbar named "Edit" with 3 buttons: Cut, Copy, Paste
if tb.create("Edit") then
   tb.addButton(ToolbarEditCluster, ToolbarButtonPush,
                MenuEditCut, BitmapEditCut, "Cut")


   tb.addButton(ToolbarEditCluster, ToolbarButtonPush,
                MenuEditCopy, BitmapEditCopy, "Copy")


   tb.addButton(ToolbarEditCluster, ToolbarButtonPush,
                MenuEditPaste, BitmapEditPaste, "Paste")


endif
endMethod
```

The following example creates a Toolbar named File and adds three buttons using Corel Paradox constants. A fourth button is added using a custom graphic object.

```
method pushButton (var eventInfo Event)
var
   tb  Toolbar
   gr  graphic
endvar


if tb.create("File") then
   tb.addButton(ToolbarFileCluster, ToolbarButtonPush,
                MenuTableOpen, BitmapOpenTable, "Open Table")


   tb.addButton(ToolbarFileCluster, ToolbarButtonPush,
                MenuFormOpen, BitmapOpenForm, "Open Form")


   tb.addButton(ToolbarFileCluster, ToolbarButtonPush,
                MenuReportOpen, BitmapOpenReport, "Open Report")


   ;// Add a button with a custom bitmap (pick a valid name)
   gr.readFromFile("Alias.bmp")
   tb.addButton(ToolbarModeCluster, ToolbarButtonPush,
                MenuFileAliases, gr, "Alias")
endif
endMethod
```

attach method

Binds a Toolbar type to an existing Toolbar.

Syntax

```
attach ( const toolbarName String ) Logical
```

Description

attach binds a Toolbar type to an existing Toolbar using the name specified in *toolbarName*. The reserved name Standard can be used to attach to the Corel Paradox Toolbar.

You can access a Toolbar by attaching to an existing toolbar or creating a new one.

Example

{button ,AL(`OPAL_TYPE_TOOLBAR;OPAL_METH_TOUNATTACH;OPAL_METH_TOCREATE;OPAL_METH_TOCREATETABBED;',0,"Defaultoverview",)} Related Topics

attach example

The following example attaches a Toolbar named MyToolbar. This code assumes that the Toolbar already exists:

```
method pushbutton (var eventInfo Event)
var
   tbar      Toolbar
endvar


   if tbar.attach("Standard") then
      msginfo("Attach", "Successful")
   else
      msginfo("Attach", "Failed")
   endif
endMethod
```

create method

Creates a Toolbar.

Syntax

```
create ( const toolbarName String [, const parentToolbarName String ]) Logical
```

Description

create creates a Toolbar specified by *toolbarName*. *toolbarName* is used in the caption when the Toolbar is floating. The name cannot be Standard, which is reserved for the Corel Paradox Toolbar.

parentToolbarName is the name of the tabbed parent Toolbar of the new Toolbar.

You can access a Toolbar by attaching to an existing toolbar or creating a new one.

Example

{button ,AL(`OPAL_TYPE_TOOLBAR;OPAL_METH_TOATTACH;OPAL_METH_TOUNATTACH;OPAL_METH_TOCREATETABBED;',0,"Defaultoverview",)} Related Topics

create example

The following example uses **createTabbed** to create a tabbed Toolbar named Test. Test is created using two Toolbars created with the **create** method (the Edit and File Toolbars).

```
method pushButton (var eventInfo Event)
var
   tbTabbed  Toolbar
   tbEdit    Toolbar
   tbFile    Toolbar
endvar


;// Create a tabbed Toolbar named "Test"
;// that will be composed of two Toolbars:
;// "Edit" and "File"
if tbTabbed.createTabbed("Test") then
   ; Create a Toolbar named "Edit" with 3 buttons: Cut, Copy, Paste
   if tbEdit.create("Edit", "Test") then
      tbEdit.addButton(ToolbarEditCluster, ToolbarButtonPush,
                       MenuEditCut, BitmapEditCut, "Cut")


      tbEdit.addButton(ToolbarEditCluster, ToolbarButtonPush,
                       MenuEditCopy, BitmapEditCopy, "Copy")


      tbEdit.addButton(ToolbarEditCluster, ToolbarButtonPush,
                       MenuEditPaste, BitmapEditPaste, "Paste")
   endif


   if tbFile.create("File", "Test") then
      tbFile.addButton(ToolbarFileCluster, ToolbarButtonPush,
                       MenuTableOpen, BitmapOpenTable, "Open Table")


      tbFile.addButton(ToolbarFileCluster, ToolbarButtonPush,
                       MenuFormOpen, BitmapOpenForm, "Open Form")


      tbFile.addButton(ToolbarFileCluster, ToolbarButtonPush,
                       MenuReportOpen, BitmapOpenReport, "Open Report")
   endif
endif
endMethod
```

createTabbed method

Creates a tabbed Toolbar.

Syntax

```
createTabbed ( const toolbarName String ) Logical
```

Description

createTabbed creates a tabbed Toolbar named *toolbarName. toolbarName* is used in the caption when the Toolbar is floating. The name cannot be Standard, which is reserved for the Corel Paradox Toolbar.

Example

{button ,AL(`OPAL_TYPE_TOOLBAR;OPAL_METH_TOATTACH;OPAL_METH_TOUNATTACH;OPAL_METH_TOCREATE;',0,"Defaultoverview",)} **Related Topics**

createTabbed example

The following example uses **createTabbed** to create a tabbed Toolbar named Test. Test is created using two Toolbars created with the **create** method (the Edit and File Toolbars).

```
method pushButton (var eventInfo Event)
var
   tbTabbed  Toolbar
   tbEdit    Toolbar
   tbFile    Toolbar
endvar


;// Create a tabbed Toolbar named "Test"
;// that will be composed of two Toolbars:
;// "Edit" and "File"
if tbTabbed.createTabbed("Test") then
   ; Create a Toolbar named "Edit" with 3 buttons: Cut, Copy, Paste
   if tbEdit.create("Edit", "Test") then
      tbEdit.addButton(ToolbarEditCluster, ToolbarButtonPush,
                       MenuEditCut,BitmapEditCut, "Cut")


      tbEdit.addButton(ToolbarEditCluster, ToolbarButtonPush,
                       MenuEditCopy,BitmapEditCopy, "Copy")


      tbEdit.addButton(ToolbarEditCluster, ToolbarButtonPush,
                       MenuEditPaste,BitmapEditPaste, "Paste")
   endif


   if tbFile.create("File", "Test") then
      tbFile.addButton(ToolbarFileCluster, ToolbarButtonPush,
                       MenuTableOpen, BitmapOpenTable, "Open Table")


      tbFile.addButton(ToolbarFileCluster, ToolbarButtonPush,
                       MenuFormOpen, BitmapOpenForm, "Open Form")


      tbFile.addButton(ToolbarFileCluster, ToolbarButtonPush,
                       MenuReportOpen, BitmapOpenReport, "Open Report")
   endif
endif
endMethod
```

empty method

Removes the existing buttons from the Toolbar.

Syntax

```
empty ( ) Logical
```

Description

empty removes the existing buttons from the attached Toolbar. **empty** returns True if the Toolbar is successfully emptied.

Example

{button ,AL(`OPAL_TYPE_TOOLBAR;OPAL_METH_TOADDBUTTON;OPAL_METH_TOHIDE;',0,"Defaultoverview",)} **Related Topics**

empty example

The following example attaches a Toolbar named MyToolbar and removes the Toolbar's buttons using **empty**. If the attach fails, an "Unable to attach" message appears.

```
method pushbutton (var eventInfo Event)
var
   tbar      Toolbar
endvar


if tbar.attach("MyToolbar") then
   tbar.empty()
else
   msgInfo("Toolbar error", "Unable to attach.")
endif


endMethod
```

getPosition method

Returns the position of a floating Toolbar.

Syntax

```
getPosition ( var x LongInt, var y LongInt ) Logical
```

Description

getPosition returns the position of a floating Toolbar. The Toolbar's coordinates are specified in pixels, relative to the top-left corner of the screen.

Example

{button ,AL(`OPAL_TYPE_TOOLBAR;OPAL_METH_TOGETSTATE;OPAL_METH_TOSETPOSITION;OPAL_METH_TOSETSTATE;',0,"Defaultoverview",)} Related Topics

getPosition example

The following example displays the X and Y coordinates of the attached Toolbar named MyToolbar:

```
method pushbutton (var eventInfo Event)
var
   liX, liY  LongInt
   tbar      Toolbar
endvar


if tbar.attach("MyToolbar") then
   tbar.getPosition(liX, liY)
   liX.view("X coordinate")
   liY.view("Y coordinate")
endif
endMethod
```

getState method

Retrieves the Toolbar's current state.

Syntax

```
getState ( ) smallInt
```

Description

getState retrieves the Toolbar's current state. **getState** returns True if the Toolbar state is successfully retrieved.

Example

{button ,AL(`OPAL_TYPE_TOOLBAR;OPAL_METH_TOGETPOSITION;OPAL_METH_TOSETPOSITION;OPAL_METH_TOSETSTATE;',0,"Defaultoverview",)} Related Topics

getState example

The following example displays the current state of a Toolbar named MyToolbar. If this code cannot attach to MyToolbar, an "Unable to attach" message appears.

```
method pushbutton (var eventInfo Event)
var
   tbar      Toolbar
endvar


if tbar.attach("MyToolbar") then
   msgInfo("MyToolbar", "Current State: " + String(tbar.getState()))
else
   msgInfo("Toolbar error", "Unable to attach.")
endif


endMethod
```

hide method

Hides a Toolbar.

Syntax

hide ( ) Logical

Description

hide hides a Toolbar. **hide** returns True if the Toolbar is successfully hidden. **hide** performs the same function as the procedure **hideToolbar**.

Example

{button ,AL(`OPAL_TYPE_TOOLBAR;OPAL_METH_TOEMPTY;OPAL_METH_TOISVISIBLE;',0,"Defaultoverview",)} **Related Topics**

hide example

The following example hides a Toolbar named MyToolbar. If the Toolbar is not visible, this method displays it:

```
method pushbutton (var eventInfo Event)
var
   tbar  Toolbar
endvar


if tbar.attach("MyToolbar") then
   if tbar.isVisible() then
      tbar.hide()
   else
      tbar.show()
   endif
endif


endMethod
```

isVisible method

Determines whether the specified Toolbar is visible.

Syntax

```
isVisible ( ) Logical
```

Description

isVisible determines whether the specified Toolbar is visible. **isVisible** returns True if the Toolbar is visible and False if the Toolbar is hidden. This method performs the same function as the **isToolbarShowing** procedure.

Example

{button ,AL(`OPAL_TYPE_TOOLBAR;OPAL_METH_TOHIDE;OPAL_METH_TOSHOW;',0,"Defaultoverview",)}

Related Topics

isVisible example

The following example prints a message stating whether a Toolbar named MyToolbar is visible. If this code can not attach to MyToolbar, an "Unable to attach" message appears.

```
method pushbutton (var eventInfo Event)
var
   tbar  Toolbar
endvar


   if tbar.attach("MyToolbar") then
      if tbar.isVisible() then
         msgInfo("MyToolbar" , "Toolbar is Visible")
      else
         msgInfo("MyToolbar" , "Toolbar is not Visible")
      endif
   else
      msgInfo("Toolbar error", "Unable to attach.")
   endif
endMethod
```

remove method

Removes the specified Toolbar from the screen.

Syntax

```
remove ( ) Logical
```

Description

remove removes the specified Toolbar from the screen. **remove** returns True if the Toolbar was successfully removed; otherwise it returns False.

Example

{button ,AL(`OPAL_TYPE_TOOLBAR;OPAL_METH_TOEMPTY;OPAL_METH_TOHIDE;',0,"Defaultoverview",)}

Related Topics

remove example

The following example removes a Toolbar named MyToolbar from the screen. If this code can not attach to MyToolbar, an "Unable to attach" message appears.

```
method pushbutton (var eventInfo Event)
var
   tbar  Toolbar
endvar


if tbar.attach("MyToolbar") then
   tbar.remove()
else
   msgInfo("Toolbar error", "Unable to attach.")
endif


endMethod
```

removeButton method

Removes a button from the Toolbar.

Syntax

```
removeButton ( const idCluster SmallInt, const idNum SmallInt ) Logical
```

Description

removeButton removes a button from the Toolbar using the host cluster and the position in the cluster. The cluster is specified with *idCluster* and the position of the button in the cluster (from left to right starting at 0) is specified by *idNum*. **removeButton** returns True if the button is successfully removed.

Example

{button ,AL(`OPAL_TYPE_TOOLBAR;OPAL_METH_TOADDBUTTON;OPAL_METH_TOEMPTY;',0,"Defaultoverview",)} Related Topics

removeButton example

The following example removes the a button from a Toolbar named MyToolbar. Because *idCluster* and *idNum* start with zero, this example removes the third button from the second cluster. If this code can not attach to MyToolbar, an "Unable to attach" message appears.

```
method pushbutton (var eventInfo Event)
var
   tbar  Toolbar
endvar


if tbar.attach("MyToolbar") then
   tbar.removebutton(1,2)       ;//idcluster=1, the 2nd from left
                                ;//idnum=2, the 3rd from left
else
   msgInfo("Toolbar error", "Unable to attach.")
endif


endMethod
```

setPosition method

Changes the position of a floating Toolbar.

Syntax

```
setPosition ( const x LongInt, const y LongInt ) Logical
```

Description

setPosition changes the position of a floating Toolbar to the coordinates specified in *x* and *y*. The *x* and *y* coordinates are specified in pixels and relative to the upper-left corner of the screen. **setPosition** returns True if the position of the Toolbar is successfully changed.

Example

{button ,AL(`OPAL_TYPE_TOOLBAR;OPAL_METH_TOGETPOSITION;OPAL_METH_TOGETSTATE;OPAL_METH_TOSETSTATE;',0,"Defaultoverview",)} Related Topics

setPosition example

The following example changes the position of a Toolbar named MyToolbar 500 pixels to the right and 400 pixels up from its current position.

```
method pushbutton (var eventInfo Event)
var
   liX, liY  LongInt
   tbar      Toolbar
endvar


if tbar.attach("MyToolbar") then
   tbar.getPosition(liX, liY)
   view("From: " + string(liX) + ", "
                 + string(liY) +
          "To: " + string(liX + 2800) + " , "
                 + string(liY + 2800))
   tbar.setPosition(liX + 500, liY + 400)
endif


endMethod
```

setState method

Sets the state of the Toolbar.

Syntax

```
setState ( const state SmallInt ) Logical
```

Description

setState sets the Toolbar to the specified ***state***. There are six Toolbar states:

- ToolbarStateTop: docked at the top of the window
- ToolbarStateLeft: docked at the left of the window
- ToolbarStateRight: docked on the right side of the window
- ToolbarStateBottom: docked at the bottom of the window
- ToolbarStateFloatHorizontal: floating horizontally
- ToolbarStateFloatVertical: floating vertically

setState returns True if the Toolbar state is successfully set.

Example

{button ,AL(`OPAL_TYPE_TOOLBAR;OPAL_METH_TOGETPOSITION;OPAL_METH_TOGETSTATE;OPAL_METH_TOSETPOSITION;',0,"Defaultoverview",)} Related Topics

setState example

The following example displays a dialog that allows the user to set the state of the Toolbar named MyToolbar. If this code can not attach to MyToolbar, an "Unable to attach" message appears.

```
method pushbutton (var eventInfo Event)
var
   siState  SmallInt
   tbar     Toolbar
endvar


if tbar.attach("MyToolbar") then
   siState = tbar.getState()
   siState.view("Enter State: (0-7)")
   tbar.setState(siState)
else
   msgInfo("Toolbar error", "Unable to attach.")
endif


endMethod
```

show method

Shows a Toolbar.

Syntax

show ( ) Logical

Description

show shows a Toolbar.

There are five toolbars available on the Corel Paradox desktop: Standard, Global, Object, Align, and Format. The Standard toolbar is available by default and can be displayed or hidden using the **show** and **hide** methods.

This method performs the same function as the **showToolbar** procedure.

The Object and Align toolbars are used in the design environment only and are not available through ObjectPAL. To display the Global and Format toolbars using the **show** method, you must first issue the following PAL statements:

```
   ;// to display the Global toolbar
winPostMessage(ap.windowHandle(),winGetMessageID("WM_COMMAND"),30831,0)


   ;// to display the Format toolbar
winPostMessage(ap.windowHandle(),winGetMessageID("WM_COMMAND"),7903,0)
```

Example

{button ,AL(`OPAL_TYPE_TOOLBAR;OPAL_METH_TOEMPTY;OPAL_METH_TOHIDE;OPAL_METH_TOISVISIBLE;',0,"Defaultoverview",)} **Related Topics**

show example

The following example hides a Toolbar named MyToolbar. If the Toolbar is already hidden, this method displays it.

```
method pushbutton (var eventInfo Event)
var
   tbar      Toolbar
endvar


if tbar.attach("MyToolbar") then
   if tbar.isVisible() then
      tbar.hide()
   else
      tbar.show()
   endif
endif

endMethod
```

unAttach method

Removes the attachment from the Toolbar.

Syntax

unAttach ( ) Logical

Description

unAttach removes the attachment from the Toolbar.

Example

{button ,AL(`OPAL_TYPE_TOOLBAR;OPAL_METH_TOATTACH;OPAL_METH_TOCREATE;OPAL_METH_TOCREATETABBED;;',0,"Defaultoverview",)} **Related Topics**

unAttach example

The following example attaches a Toolbar named MyToolbar, sets its state, and then unattaches.

```
method pushbutton (var eventInfo Event)
var
   tbar      Toolbar
endvar


   if tbar.attach("MyToolbar") then
      tbar.setState(ToolbarStateTop)
           tbar.unattach()
   endif


endMethod
```

showApplicationBar method

Toggles the visible property of the Application Bar.

Syntax

showApplicationBar (Show Logical) Logical

Show=True	If the Application bar is not already visible, it will appear
Show=False	If the Application bar is visible, it will be hidden

Description

showApplicationBar toggles the visible property of the Application Bar.

Example

{button ,AL(`OPAL_TYPE_TOOLBAR;OPAL_METH_TOEMPTY;OPAL_METH_TOHIDE;OPAL_METH_TOISVISIBLE;',0,"Defaultoverview",)} Related Topics

showApplicationBar example

This example uses the isAppBarVisible method to check the state of the Application bar, then uses showApplcation bar to either show or hide the Applcation bar depending on its state.

```
method pushButton(var eventInfo Event)
    if isappbarvisible()=True then    ;//checks current state of application bar
        msginfo("stop","application bar is being hidden")
        Showapplicationbar(False)     ;// hides application bar
    else
        msginfo("Stop","application bar is being shown")
        showapplicationbar(True)       ;//shows application bar
    endif
endMethod
```

isAppBarVisible example

See the example for **showApplicationBar**.

isAppBarVisible method

Checks the state of the Application Bar.

Syntax

isAppBarVisible ( ) Logical

Description

isAppBarVisible checks the state of the Application bar and returns a logical.

Example

{button ,AL(`OPAL_TYPE_TOOLBAR;OPAL_METH_TOEMPTY;OPAL_METH_TOHIDE;OPAL_METH_TOISVISIBLE;',0,"Defaultoverview",)} Related Topics

Mail type

The Mail type allows you to compose electronic mail messages and transmit them using a MAPI-compliant mail system (e.g., Microsoft Mail). A Mail type variable holds a single mail message. It also holds current mail session status (set by **logon**), so that multiple mail messages can be sent (sequentially) in a single session. Declare variables of type Mail to facilitate the manipulation of mail messages, and then use the Mail methods to set (and retrieve) information about the message (such as the message subject and the recipients).

Methods for the Mail type

addAddress
addAttachment
addressBook
addressBookTo
empty
emptyAddresses
emptyAttachments
enumInbox
getAddress
getAddressCount
getAttachment
getAttachmentCount
getMessage
getMessageType
getSender
getSubject
logoff
logoffDlg
logon
logonDlg
readMessage
send
sendDlg
setMessage
setMessageType
setSubject

 **Print related ObjectPAL methods and examples**

addAddress method

Adds an addressee to a message.

Syntax

```
1. addAddress ( const address String )

2. addAddress ( const address String, const addressType SmallInt )
```

Description

addAddress adds an addressee to the message. Syntax 1 defaults to a To type addressee, Syntax 2 allows you to specify one of the MailAddressTypes Constants: MailAddrTo, MailAddrCC, or MailAddrBC. Addressees are not checked for validity until the message is sent.

Example

{button ,AL(`OPAL_TYPE_MAIL;OPAL_METH_MLADDRESSBOOK;OPAL_METH_MLADDRESSBOOKTO;OPAL_METH_MLEMPTYADDRESSES;OPAL_METH_MLGETADDRESS;OPAL_METH_MLGETADDRESSCOUNT;',0,"Defaultoverview",)} Related Topics

addAddress example

The following example sends a message (about sales results) to John Doe and copies the message to Susan Smith. It assumes the user is logged on.

```
var
   m  MAIL
endVar
method pushButton ( var eventInfo Event )
   m.addAddress("JDOE")
   m.addAddress("SSMITH", MailAddrCC)
   m.setSubject("Final sales numbers")
   m.setMessage("The final sales numbers are attached")
   m.addAttachment("SALES.TXT")
   m.send()  ; Send the message
endMethod
```

addAttachment method

Adds an attachment to the message.

Syntax

```
1. addAttachment ( const fileName String )
2. addAttachment ( const fileName String, const moniker String )
3. addAttachment ( const fileName String, const moniker String, const displayPos LongInt )
```

Description

addAttachment adds an attachment to the message. Syntax 1 sends the specified *fileName*. Syntax 2 sends the specified *fileName*, but displays the name specified in *moniker*. Some mail systems (for example, Microsoft Mail) allow the attachment icon to be displayed in the message text; in this case, you can use Syntax 3 to specify the position in the text that the file should appear. (With Microsoft mail, if you specify one, the first character of the message to be displaced by the icon for the specified attachment).

Some mail systems place limits on the number, size, and/or type of attachments you can use (a few mail systems still don't support binary attachments). No attempt is made to verify the existence of the files until the message is sent. Aliases can be used to specify attachment names.

Example

{button ,AL(`OPAL_TYPE_MAIL;OPAL_METH_MLEMPTYATTACHMENTS;OPAL_METH_MLGETATTACHMENT;OPAL_METH_MLGETATTACHMENTCOUNT;',0,"Defaultoverview",)} Related Topics

addAttachment example

The following example sends a message (about sales results) to John Doe and copies the message to Susan Smith. It assumes the user is logged on.

```
var
   m  MAIL
endVar
method pushButton ( var eventInfo Event )
   m.addAddress("JDOE")
   m.addAddress("SSMITH", MailAddrCC)
   m.setSubject("Final sales numbers")
   m.setMessage("The final sales numbers are attached")
   m.addAttachment("SALES.TXT")
   m.send()  ; Send the message
endMethod
```

addressBook method

Displays the address book.

Syntax

```
1. addressBook ( )

2. addressBook ( const numberOfLists SmallInt )
```

Description

addressBook displays the address book and allows the user to modify the list of addressees. Syntax 1 allows all types of addressees (To, CC, BC) to be updated. Syntax 2 allows you to limit the number of address lists to be updated: *numberOfLists* = 1 shows only the To addressees, *numberOfLists* = 2 shows the To and CC addressees, *numberOfLists* = 3 shows the To, CC, and BC addressees.

If an existing mail session is not active, the user may be prompted with a logon dialog box. Use the **logon** method to create a mail session.

Example

{button ,AL(`OPAL_TYPE_MAIL;OPAL_METH_MLADDADDRESS;OPAL_METH_MLADDRESSBOOKTO;OPAL_METH_MLEMPTYADDRESSES;OPAL_METH_MLGETADDRESS;OPAL_METH_MLGETADDRESSCOUNT;',0,"Default overview",)} Related Topics

addressBook example

The following example updates a distribution list kept in a table:

```
method pushButton ( var eventInfo Event )
   var m MAIL tc TCURSOR idx LONGINT address STRING addrtype SMALLINT endVar
   tc.open("distribution list.db")
   scan tc:  ; read the address list
      m.addAddress( tc."Addressee" )
   endscan
   m.addressBook( 1 ) ; Display the list for editing
   tc.edit( )
   tc.empty( ) ; clear the old list
   for idx from 1 to m.getAddressCount( ) ; write out the new list
       tc.insertRecord( )
       m.getAddress( idx, address, addrtype )
       tc."Addressee" = address
       tc.unlockRecord( )
   endfor
   tc.close( )
endMethod
```

addressBookTo method

Displays the To list from the address book.

Syntax

```
addressBookTo ( const prompt String )
```

Description

addressBookTo displays the To list from the address book and allows the user to modify the list of addressees. **addressBookTo** displays only the To list, but allows you to override what the list is called (e.g., Routing).

If an existing mail session is not active, the user may be prompted with a logon dialog box. Use the **logon** method to create a mail session.

Example

{button ,AL(`OPAL_TYPE_MAIL;OPAL_METH_MLADDADDRESS;OPAL_METH_MLADDRESSBOOK;OPAL_METH_MLEMPTYADDRESSES;OPAL_METH_MLGETADDRESS;OPAL_METH_MLGETADDRESSCOUNT;',0,"Defaultoverview",)} Related Topics

addressBookTo example

The following example allows the user to update a distribution list kept in a table:

```
method pushButton ( var eventInfo Event )
   var m MAIL tc TCURSOR idx LONGINT address STRING addrtype SMALLINT endVar
   tc.open("distribution list.db")
   scan tc:  ; read the address list
      m.addAddress( tc."Addressee" )
   endscan
   m.addressBookTo( "Fundraiser Mail List" )
   tc.edit( )
   tc.empty( ) ; clear the old list
   for idx from 1 to m.getAddressCount( ) ; write out the new list
       tc.insertRecord( )
       m.getAddress( idx, address, addrtype )
       tc."Addressee" = address
       tc.unlockRecord( )
   endfor
   tc.close( )
endMethod
```

empty method

Empties the contents of the mail variable.

Syntax

```
empty ( )
```

Description

empty empties the contents of the mail variable (clears the message). The session (which is set by the **logon** method), if any, is unaffected.

Example

{button ,AL(`OPAL_TYPE_MAIL;OPAL_METH_MLEMPTYADDRESSES;OPAL_METH_MLEMPTYATTACHMENTS;',0,"Defaultoverview",)} Related Topics

empty example

The following example sends a message (about sales results) to John Doe, copies the message to Susan Smith and then sends a different message to Bill Brown. It assumes the user is logged on.

```
var
   m  MAIL
endVar
method pushButton ( var eventInfo Event )
   m.addAddress("JDOE")
   m.addAddress("SSMITH", MailAddrCC)
   m.setSubject("Final sales numbers")
   m.setMessage("The final sales numbers are attached")
   m.addAttachment("SALES.TXT")
   m.send()  ; Send the message

   m.empty()  ; Clear out the old message

   m.addAddress("BBROWN")
   m.setSubject("Final sales numbers sent")
   m.setMessage("Bill, John and Susan have the final sales now")
   m.send()  ; Send the message
endMethod
```

emptyAddresses method

Deletes all the addresses attached to a message.

Syntax

```
emptyAddresses ( )
```

Description

emptyAddresses sets the number of addresses attached to the message to zero.

Example

{button ,AL(`OPAL_TYPE_MAIL;OPAL_METH_MLADDADDRESS;OPAL_METH_MLADDRESSBOOK;OPAL_METH_MLADDRESSBOOKTO;OPAL_METH_MLEMPTY;OPAL_METH_MLEMPTYADDRESSES;OPAL_METH_MLGETADDRESS;OPAL_METH_MLGETADDRESSCOUNT;',0,"Defaultoverview",)} Related Topics

emptyAddresses example

The following example sends a message (about sales results) to John Doe, copies the message to Susan Smith, and sends a different message to Bill Brown. It assumes the user is logged on.

```
var
   m  MAIL
endVar
method pushButton ( var eventInfo Event )
   m.addAddress("JDOE")
   m.addAddress("SSMITH", MailAddrCC)
   m.setSubject("Final sales numbers")
   m.setMessage("The final sales numbers are attached")
   m.addAttachment("SALES.TXT")
   m.send()  ; Send the message

   m.emptyAddresses()  ; Clear out the old Addresses

   m.addAddress("BBROWN")
   m.setMessage("Bill, John and Susan have the final sales now")
   m.send()  ; Send with subject & attachment specified earlier
endMethod
```

emptyAttachments method

Deletes all the attachments to a message.

Syntax

```
emptyAttachments ( )
```

Description

emptyAttachments sets the number of attachments to the message to zero.

Example

{button ,AL(`OPAL_TYPE_MAIL;OPAL_METH_MLADDATTACHMENT;OPAL_METH_MLEMPTY;OPAL_METH_MLEMPTYATTACHMENTS;OPAL_METH_MLGETATTACHMENT;OPAL_METH_MLGETATTACHMENTCOUNT;',0,"Defaultoverview",)} Related Topics

emptyAttachments example

The following example sends a message (about sales results) to John Doe, copies the message to Susan Smith, and sends a different message to Bill Brown. It assumes the user is logged on.

```
method pushButton ( var eventInfo Event )
var
  m  MAIL
endVar
   m.addAddress("JDOE")
   m.addAddress("SSMITH", MailAddrCC)
   m.setSubject("Final sales numbers")
   m.setMessage("The final sales numbers are attached")
   m.addAttachment("SALES.TXT")
   m.send()  ; Send the message


   m.emptyAddresses()  ; Clear out the old Addressee's
   m.emptyAttachments()  ; Clear out the old Attachment


   m.addAddress("BBROWN")
   m.setMessage("Bill, John and Susan have the final sales now")
   m.send()  ; Send with subject specified earlier
endMethod
```

enumInbox method

Fills an array with the list of messages in the in box.

Syntax

```
enumInbox ( var ids Array [] String, const unreadOnly Logical, [ const seedId String, const
maxCount LongInt, [ const msgType String ] ] )
```

Description

enumInbox fills the array specified by *ids* with the IDs of messages in the in box. *unreadonly* is a True or False value that indicates whether to include only unread messages.

The optional value *seedId* lets you control the starting point from which messages are listed in the array. To retrieve the first set of messages, use a blank string (""). To retrieve subsequent sets of messages, use the last ID read from the previous set. If you specify a value for *seedId*, you must also specify a value for *maxCount*. *maxCount* lets you control the maximum number of messages retrieved in a set.

msgType lets you specify the type of message. Message types are mail system dependent consult your mail system documentation for information on the message types it supports.

Example

{button ,AL(`OPAL_TYPE_MAIL;OPAL_METH_MLADDRESSBOOK;opal_meth_mlreadMessage;',0,"Defaultoverview",)} Related Topics

enumInbox example

The following example gets the list of message IDs for unread messages in the in box. The example reads the first set of messages and stops after listing a maximum of 100 messages. A custom method *ProcessMessage* is called to process each message until there are no messages left in the set. If additional messages remain unread and need processing, the loop repeats. When there are no more messages to process, the method ends. This example assumes the user is logged on.

```
method pushButton ( var eventInfo Event )
var
   msg         Mail
   inboxIds    Array [] String
   seedId      String
   i           LongInt
endVar
  seedId = ""                              ; set to retrieve first message
  while True                               ; Process all unread messages
      msg.enumInbox( inboxIds, True, seedId, 100)
      for i from 1 to inboxIds.size()
          processMessage(inboxIds[i])      ; run a custom method to process
                                           ; each message
      endFor
      if inboxIds.size() < 100 then
          quitloop
      endIf
      seedId = inboxIds[100]               ; set seed to last message read
  endWhile
endMethod
```

getAddress method

Retrieves the specified addressee information.

Syntax

```
1. getAddress ( const index LongInt, var address String, var addressType SmallInt )

2. getAddress ( const index LongInt, var address String, var fullAddress String, var
addressType SmallInt )
```

Description

getAddress retrieves the specified addressee information, where *index* is between 1 and **getAddressCount**, inclusive. *addressType* is one of the MailAddressTypes Constants: MailAddrTo, MailAddrCC, or MailAddrBC.

In addition to the above information, Syntax 2 retrieves the full address name of the addressee and stores this value in *fullAddress*. Full address information is available only after a MAPI-compliant mail system has made this information available to Corel Paradox. A blank value for *fullAddress* indicates that the MAPI-compliant mail system has not yet provided this information.

Example

{button ,AL(`OPAL_TYPE_MAIL;OPAL_METH_MLADDADDRESS;OPAL_METH_MLADDRESSBOOK;OPAL_METH_MLADDRESSBOOKTO;OPAL_METH_MLEMPTYADDRESSES;OPAL_METH_MLGETADDRESSCOUNT;',0,"Defaultoverview",)} Related Topics

getAddress example

The following example allows the user to update a distribution list kept in a table:

```
method pushButton ( var eventInfo Event )
   var m MAIL tc TCURSOR idx LONGINT address STRING addrtype SMALLINT endVar
   tc.open("distribution list.db")
   scan tc:  ; read the address list
      m.addAddress( tc."Addressee" )
   endscan
   m.addressBookTo( "Fundraiser Mail List" )
   tc.edit( )
   tc.empty( ) ; clear the old list
   for idx from 1 to m.getAddressCount( ) ; write out the new list
       tc.insertRecord( )
       m.getAddress( idx, address, addrtype )
       tc."Addressee" = address
       tc.unlockRecord( )
   endfor
   tc.close( )
endMethod
```

getAddressCount method

Returns the number of addressees attached to the current message.

Syntax

getAddressCount ( ) LongInt

Description

getAddressCount returns the number of addressees attached to the current message.

Example

{button ,AL(`OPAL_TYPE_MAIL;OPAL_METH_MLADDADDRESS;OPAL_METH_MLADDRESSBOOK;OPAL_METH_MLADDRESSBOOKTO;OPAL_METH_MLEMPTYADDRESSES;OPAL_METH_MLGETADDRESS;',0,"Defaultoverview",)} Related Topics

getAddressCount example

The following example allows the user to update a distribution list kept in a table:

```
method pushButton ( var eventInfo Event )
   var m MAIL tc TCURSOR idx LONGINT address STRING addrtype SMALLINT endVar
   tc.open("distribution list.db")
   scan tc:  ; read the address list
      m.addAddress( tc."Addressee" )
   endscan
   m.addressBookTo( "Fundraiser Mail List" )
   tc.edit( )
   tc.empty( ) ; clear the old list
   for idx from 1 to m.getAddressCount( ) ; write out the new list
       tc.insertRecord( )
       m.getAddress( idx, address, addrtype )
       tc."Addressee" = address
       tc.unlockRecord( )
   endfor
   tc.close( )
endMethod
```

getAttachment method

Retrieves specific attachment information.

Syntax

```
getAttachment ( const index LongInt, var fileName String, var moniker String, var displayPos LongInt )
```

Description

getAttachment retrieves the attachment information for the attachment specified by *index*. *index* is a number between 1 and **getAttachmentCount**, inclusive. *filename*, *moniker*, and *displayPos* are variables whose values are filled in by this method. *filename* represents the name of the attachment file. *moniker* is the name displayed in the MAPI mail dialog (defaults to the filename). *displayPos* is the display position of the attachment's icon in the MAPI mail dialog.

Example

{button ,AL(`OPAL_TYPE_MAIL;OPAL_METH_MLADDATTACHMENT;OPAL_METH_MLEMPTYATTACHMENTS; OPAL_METH_MLGETATTACHMENTCOUNT;',0,"Defaultoverview",)} Related Topics

getAttachment example

The following example gets the list of attachments from a mail variable. The mail variable *m* and its attachments are presumed to have been defined and added elsewhere. The example assumes the user is logged on.

```
method pushButton ( var eventInfo Event )
   var
       list DYNARRAY [] STRING
       indx LONGINT
       filename STRING
       moniker STRING
       pos LONGINT
   endVar
   for indx from 1 to m.getAttachmentCount()
       m.getAttachment(indx, filename, moniker, pos)
       list[indx]=filename
   endfor
   list.view("attachments:")
endMethod
```

getAttachmentCount method

Returns the number of attachments to the current message.

Syntax

```
getAttachmentCount ( ) LongInt
```

Description

getAttachmentCount returns the number of attachments to the current message.

Example

{button ,AL(`OPAL_TYPE_MAIL;OPAL_METH_MLADDATTACHMENT;OPAL_METH_MLEMPTYATTACHMENTS; OPAL_METH_MLGETATTACHMENT;',0,"Defaultoverview",)} Related Topics

getAttachmentCount example

The following example displays the number of attachments. The mail variable *m* and its attachments are presumed to have been defined and added elsewhere. The message assumes the user is logged on.

```
method pushButton ( var eventInfo Event )
   var

      cnt longint

   endVar
   m.addAttachment( "SALES.TXT" )
   cnt = m.getAttachmentCount( )
   cnt.view( "Number of attachments" )
endMethod
```

getMessage method

Returns the current text of the message.

Syntax

```
getMessage ( ) String
```

Description

getMessage returns the current text of the message.

Example

{button ,AL(`OPAL_TYPE_MAIL;OPAL_METH_MLEMPTY;OPAL_METH_MLGETMESSAGETYPE;OPAL_METH_MLSETMESSAGE;OPAL_METH_MLSETMESSAGETYPE;',0,"Defaultoverview",)} Related Topics

getMessage example

The following example displays the (previously set) message text. It assumes the user is logged on.

```
var
   m  MAIL
endVar
method pushButton ( var eventInfo Event )
   var msgtext string endVar
   msgtext = m.getMessage( )
   msgtext.view( "Message text" )
endMethod
```

getMessageType method

Returns the current message type.

Syntax

```
getMessageType ( ) String
```

Description

getMessageType returns the current message type. Message types are mail system dependent consult your mail system documentation for information on the message types it supports.

Example

{button ,AL(`OPAL_TYPE_MAIL;OPAL_METH_MLEMPTY;OPAL_METH_MLGETMESSAGE;OPAL_METH_MLSETMESSAGE;OPAL_METH_MLSETMESSAGETYPE;',0,"Defaultoverview",)} **Related Topics**

getMessageType example

The following example displays the (previously set) message type. It assumes the user is logged on.

```
var
   m  MAIL
endVar
method pushButton ( var eventInfo Event )
   var msgtype string endVar
   msgtype = m.getMessageType( )
   msgtype.view( "Message type" )
endMethod
```

getSender method

Returns the sender for the current message.

Syntax

```
1. getSender ( var address String )
2. getSender ( var address String, var fullAddress String )
```

Description

getSender returns the sender of the current mail message as *address*. In Syntax 2, **getSender** returns the full address of the sender as *fullAddress*. (Many mail systems differentiate between nickname addresses and the full email address. If the mail system that you use does not differentiate, *address* and *fullAddress* will return the same value.)

The sender's address (and full address) is available only when messages are read. The values will be blank for messages you are composing.

Example

{button ,AL(`OPAL_TYPE_MAIL;OPAL_METH_MLGETMESSAGE;OPAL_METH_MLGETSUBJECT;',0,"Defaultoverview",)} **Related Topics**

getSender example

The following example displays the sender information for the mail message *msg.* **getSender** and **getSubject** are called to get the subject title and sender name to create the reply mail. The message assumes the user is logged on.

```
method pushButton ( var eventInfo Event )
var
   msg,
   replyMsg       Mail
   sender,
   fullAddress    String
endVar

msg.readMessage("1234")                          ; 1234 is a valid message ID
msg.getSender(sender,fullAddress)                ; supplies sender info

replyMsg.addAddress(fullAddress)                 ; add sender to reply
replyMsg.setSubject(msg.getSubject()+" - Reply") ; set reply message
                                                 ; subject to original
                                                 ; subject plus "reply"
replyMsg.sendDlg()                               ; open the send dialog to
                                                 ; let user type contents
                                                 ; for the reply message
endMethod
```

getSubject method

Returns the current subject of the message.

Syntax

```
getSubject ( ) String
```

Description

getSubject returns the current subject of the message.

Example

{button ,AL(`OPAL_TYPE_MAIL;OPAL_METH_MLSETMESSAGE;OPAL_METH_MLSETSUBJECT;',0,"Defaultoverview",)} Related Topics

getSubject example

The following example displays the (previously set) subject for the mail variable *m* (assigned elsewhere). It assumes the user is logged on.

```
method pushButton ( var eventInfo Event )
   var

      subject string

   endVar
   subject = m.getsubject( )
   subject.view( "Subject" )
endMethod
```

logoff method

Attempts to logoff the mail system.

Syntax

```
logoff ( )
```

Description

logoff attempts to logoff the mail system without user intervention and to terminate the mail session created by **logon**. Any errors will trigger an exception.

Example

{button ,AL(`OPAL_TYPE_MAIL;OPAL_METH_MLLOGOFFDLG;OPAL_METH_MLLOGON;OPAL_METH_MLLOGONDLG;',0,"Defaultoverview",)} Related Topics

logoff example

The following example logs on, displays the send dialog box and then logs off:

```
var
   m  MAIL
endVar
method pushButton ( var eventInfo Event )
   m.logon("mypassword", "special" )
   m.sendDlg( )
   m.logoff( )
endMethod
```

logoffDlg method

Attempts to logoff the mail system with user interaction.

Syntax

```
logoffDlg ( ) Logical
```

Description

logoffDlg attempts to logoff the mail system and to terminate the mail session created by **logon**. If supported by the mail system, the user is prompted to enter logoff information, otherwise a straight logoff is done.

logoffDlg returns True if the user logs off, and False if the user cancels. Any errors will trigger an exception.

Example

{button ,AL(`OPAL_TYPE_MAIL;OPAL_METH_MLLOGOFF;OPAL_METH_MLLOGON;OPAL_METH_MLLOGONDLG;',0,"Defaultoverview",)} Related Topics

logoffDlg example

The following example logs on, displays the send dialog box, logs off and displays a logoff dialog box if appropriate:

```
var
   m  MAIL
endVar
method pushButton ( var eventInfo Event )
   m.logon("mypassword", "special" )
   m.sendDlg( )
   m.logoffDlg( )
endMethod
```

logon method

Attempts to logon to the mail system.

Syntax

```
logon ( const password String, const profileName String )
```

Description

logon attempts to logon to the mail system without user intervention. Any errors will trigger an exception.

The *password* argument is an input parameter that specifies a credential string (maximum 256 characters). If the messaging system does not require password credentials, or if it requires that the user actively enter them, *password* should be blank. When the user must enter credentials, use **logonDlg**.

The argument *profileName* is an input parameter that specifies a named profile string (maximum of 256 characters). This is the profile to use when logging on. Some mail providers accept a null *profileName* as specifying the default profile. If you don't know the *profileName*, use **LogonDlg**.

Example

{button ,AL(`OPAL_TYPE_MAIL;OPAL_METH_MLLOGOFF;OPAL_METH_MLLOGOFFDLG;OPAL_METH_MLLOGONDLG;',0,"Defaultoverview",)} Related Topics

logon example

The following example sends a message (about sales results) to John Doe, copies the message to Susan Smith, and sends a different message to Bill Brown. It uses logon to specify a special mail session and to group everything together.

```
var
   m  MAIL
endVar
method pushButton ( var eventInfo Event )
   m.logon("mypassword", "special" )
   m.addAddress("JDOE")
   m.addAddress("SSMITH", MailAddrCC)
   m.setSubject("Final sales numbers")
   m.setMessage("The final sales numbers are attached")
   m.addAttachment("SALES.TXT")
   m.send()  ; Send the message


   m.empty()  ; Clear out the old message


   m.addAddress("BBROWN")
   m.setSubject("Final sales numbers sent")
   m.setMessage("Bill, John and Susan have the final sales now")
   m.send()  ; Send the message
   m.logoff()
endMethod
```

logonDlg method

Attempts to logon to the mail system with user interaction.

Syntax

```
1. logonDlg ( ) Logical
2. logonDlg ( const password String, const profile String ) Logical
```

Description

logonDlg attempts to logon to the mail system with user interaction. If necessary, the user is prompted to enter logon information. If successful, a mail session is created. The session stays active until the **logoff** method is called, or the mail variable goes out of scope.

logonDlg returns True if the user logs on, and False if the user cancels. Any errors will trigger an exception.

Example

{button ,AL(`OPAL_TYPE_MAIL;OPAL_METH_MLLOGOFF;OPAL_METH_MLLOGOFFDLG;OPAL_METH_MLLOGON;',0,"Defaultoverview",)} Related Topics

logonDlg example

The following example sends a message (about sales results) to John Doe, copies the message to Susan Smith, and sends a different message to Bill Brown. It uses logonDlg so that the user will only have to specify a mail password once.

```
var
   m  MAIL
endVar
method pushButton ( var eventInfo Event )
   m.logonDlg( )
   m.addAddress("JDOE")
   m.addAddress("SSMITH", MailAddrCC)
   m.setSubject("Final sales numbers")
   m.setMessage("The final sales numbers are attached")
   m.addAttachment("SALES.TXT")
   m.send()  ; Send the message


   m.empty()  ; Clear out the old message


   m.addAddress("BBROWN")
   m.setSubject("Final sales numbers sent")
   m.setMessage("Bill, John and Susan have the final sales now")
   m.send()  ; Send the message
   m.logoff()
endMethod
```

readMessage method

Reads a mail message.

Syntax

```
readMessage ( var messageId AnyType, [ const readOpts Anytype ] )
```

Description

readMessage reads a mail message into a Mail variable. Use MailReadOptions constants to specify reading options (multiple MailReadOptions constants may be used at the same time by adding them together.)

Example

{button ,AL(`OPAL_TYPE_MAIL;OPAL_METH_MLSENDDLG;',0,"Defaultoverview",)} **Related Topics**

readMessage example

In the following example, the in box of the MAPI-compliant mail system is enumerated to the array *InboxIds.* Each message in *InboxIds* is read and calls a custom method that contains a message processing routine. The loop repeats as many times as there are messages to process. The example assumes the user is logged on.

```
method pushButton ( var eventInfo Event )
var
   msg         Mail
   inboxIds    Array [] String
   i           LongInt
endVar
msg.enumInbox( inboxIds, True)
for i from 1 to inboxIds.size()
  msg.readMessage(inboxIds[i])
  doProcess()                       ; calls a custom method for processing
endFor
endMethod
```

send method

Sends a mail message.

Syntax

```
send ( )
```

Description

send sends a mail message without user interaction. At least one addressee must have been defined. Most mail systems require that some additional information is defined (for example, the subject).

If an existing mail session is not active, the user may be prompted with a logon dialog box. Use the **logon** method to create a mail session. Some mail provider systems may require an explicit logon call before a send, others may not.

 **Example**

{button ,AL(`OPAL_TYPE_MAIL;OPAL_METH_MLSENDDLG;',0,"Defaultoverview",)} **Related Topics**

send example

The following example sends a message (about sales results) to John Doe and copies the message to Susan Smith. It assumes the user is logged on.

```
method pushButton ( var eventInfo Event )
var
   m  MAIL
endVar
   m.addAddress("JDOE")
   m.addAddress("SSMITH", MailAddrCC)
   m.setSubject("Final sales numbers")
   m.setMessage("The final sales numbers are attached")
   m.addAttachment("SALES.TXT")
   m.send()  ; Send the message
endMethod
```

sendDlg method

Sends a mail message with user interaction.

Syntax

sendDlg ( ) Logical

Description

sendDlg sends a mail message with user interaction. The user will be shown the message as it currently exists (using the user's default MAPI mail system provider). They can then modify it before sending it.

If an existing mail session is not active, the user may be prompted with a logon dialog box. Use the **logon** method to create a mail session.

sendDlg returns True if the user sends the message, and False if they cancel. Any errors will trigger an exception.

sendDlg returns True if the user cancels the logon dialog box.

Example

{button ,AL(`OPAL_TYPE_MAIL;OPAL_METH_MLSEND;',0,"Defaultoverview",)} **Related Topics**

sendDlg example

The following example simply displays a mail dialog box for the user to enter a message. It assumes the user is logged on.

```
method pushButton ( var eventInfo Event )
var
   m    MAIL
endVar
   m.sendDlg()
endMethod
```

The following example sends a message (about sales results) to John Doe and copies the message to Susan Smith:

```
method pushButton ( var eventInfo Event )
var
   m  MAIL
endVar
   m.addAddress("JDOE")
   m.addAddress("SSMITH", MailAddrCC)
   m.setSubject("Final sales numbers")
   m.setMessage("The final sales numbers are attached")
   m.addAttachment("SALES.TXT")
   m.sendDlg()  ; Display the message so the user can edit before sending
endMethod
```

setMessage method

Sets the text of the message.

Syntax

```
setMessage ( const message String )
```

Description

setMessage sets the text of the message to *message*. The maximum length of *message* is limited by the shorter of the mail system and ObjectPAL's maximum string length. This is typically at least 32,000 characters.

Example

{button ,AL(`OPAL_TYPE_MAIL;OPAL_METH_MLEMPTY;OPAL_METH_MLGETMESSAGE;OPAL_METH_MLGETMESSAGETYPE;OPAL_METH_MLSETMESSAGETYPE;OPAL_METH_MLSETSUBJECT;',0,"Defaultoverview",)}

Related Topics

setMessage example

The following example sends a message (about sales results) to John Doe and copies the message to Susan Smith. It assumes the user is logged on.

```
var
   m  MAIL
endVar
method pushButton ( var eventInfo Event )
   m.addAddress("JDOE")
   m.addAddress("SSMITH", MailAddrCC)
   m.setSubject("Final sales numbers")
   m.setMessage("The final sales numbers are attached")
   m.addAttachment("SALES.TXT")
   m.sendDlg()  ; Display the message so the user can edit before sending
endMethod
```

setMessageType method

Sets the type of the message.

Syntax

```
setMessageType ( const messageType String )
```

Description

setMessageType sets the type of the message. Some mail systems support a *messageType*. Typically, message without a specified type is assumed to be an Inter-Personal Message; whereas, typed messages can only be read by a program asking for that particular message type.

Using message types typically requires special support from your mail system. Consult your mail vendor for more information.

Example

{button ,AL(`OPAL_TYPE_MAIL;OPAL_METH_MLEMPTY;OPAL_METH_MLGETMESSAGE;OPAL_METH_MLGETMESSAGETYPE;OPAL_METH_MLGETSUBJECT;OPAL_METH_MLSETMESSAGE;OPAL_METH_MLSETSUBJECT;',0,"Defaultoverview",)} **Related Topics**

setMessageType example

The following example sends a message (about sales results) to John Doe and copies the message to Susan Smith. It uses a special message type IPM.URGENT that was previously set up on this mail system. It assumes the user is logged on.

```
var
   m  MAIL
endVar
method pushButton ( var eventInfo Event )
   m.addAddress("JDOE")
   m.addAddress("SSMITH", MailAddrCC)
   m.setSubject("Final sales numbers")
   m.setMessage("The final sales numbers are attached")
   m.addAttachment("SALES.TXT")
   m.setMessageType("IPM.URGENT")
   m.sendDlg()  ; Display the message so the user can edit before sending
endMethod
```

setSubject method

Sets the subject of the message.

Syntax

```
setSubject ( const subject String )
```

Description

setSubject sets the subject of the message to *subject*. The maximum length of *subject* is limited by the mail system. This is typically at least 80 characters.

Example

{button ,AL(`OPAL_TYPE_MAIL;OPAL_METH_MLGETSUBJECT;OPAL_METH_MLSETMESSAG;',0,"Defaultoverview",)} Related Topics

setSubject example

The following example sends a message (about sales results) to John Doe and copies the message to Susan Smith. It assumes the user is logged on.

```
var
   m  MAIL
endVar
method pushButton ( var eventInfo Event )
   m.addAddress("JDOE")
   m.addAddress("SSMITH", MailAddrCC)
   m.setSubject("Final sales numbers")
   m.setMessage("The final sales numbers are attached")
   m.addAttachment("SALES.TXT")
   m.sendDlg()  ; Display the message so the user can edit before sending
endMethod
```

Native Windows controls

Native Windows controls (NWCs) are elements of the Windows operating system that can be used in Corel Paradox. Corel Paradox uses the following NWCs:

- Combobox
- Listbox
- Progressbar
- Spinbox
- TrackBar

Accessing NWCs

NWCs are placed on a Corel Paradox form the same way as any other UIObject. The properties, methods, and events of NWCs are accessed in the same way as the properties, methods, and events of UIObjects and ActiveX controls.

The following example displays the code for a control called #spinbox3:

```
#spinbox3.propertyname = value        ; property set
value = #spinbox3.propertyname        ; property get
#spinbox3.methodname (arg, arg,...)   ;method invocation
```

NWC event handlers are written as custom methods and must exactly match the name and parameter signature of the event exposed by the control. Create and edit event handlers using Object Explorer to prevent the possibility of creating a custom method whose parameter signature does not match the event specification. If the name and parameter signature do not match exactly, the method is not an event handler.

Many methods and properties take LongInt type numeric arguments or values. To prevent the method invocation or property set from failing, cast numeric constants to type LongInt.

The following example demonstrates how to cast to the LongInt type.

```
spinbox.range (LongInt (1), LongInt (10))
spinbox.value = LongInt (1)
```

{button ,AL(`NWC_ELEMENTS;OPAL_TYPE_OLE;',0,"Defaultoverview",)} **Related Topics**
Print related ObjectPAL methods and examples

NWCs and the Value property

NWCs require the UIObject Value property. The following table lists the NWCs and their support for setting and/or getting the Value property:

NWC	Get Value ?	Set Value ?
Combobox	Yes	No
Listbox	No	Yes
Progressbar	Yes	Yes
Spinbox	Yes	Yes
TrackBar	Yes	Yes

Tips

- To get a property use the UIObject type getProperty.
- To set a property use the UIObject type setProperty.

{button ,AL(`NWC_INTRO;',0,"Defaultoverview",)} Related Topics

Combobox

Properties

Values for the Combobox properties can be specified with the Value property.

Count

Returns the number of items in the combobox (read-only).

LongInt = combobox.count

ItemIndex

Returns the index (starting at 0) of the active selection in the combobox (read-only)

LongInt = combobox.ItemIndex

Methods

addString

Adds a string to the combobox

addString (str String)

deleteString

Deletes the string that resides at the specified index (starting at 0)

deleteString (index LongInt)

findString

Returns the index (starting at 0) of the string in the combobox

LongInt findString (str String)

getText

Returns the string that resides at the specified index

string getText (index LongInt)

reset

Empties the combobox

reset ( )

Events

onSelChange

Fires when the user changes the active selection in the combobox using keyboard moves (arrow keys) or mouse actions (single click)

void onSelChange ( )

{button ,AL(`NWC_INTRO;NWC_Valueprop;',0,"Defaultoverview",)} Related Topics

Listbox

Properties

Values for the Listbox properties can be specified with the Value property.

MultiSelect

Determines whether the list box is multi-select (TRUE) or single select (FALSE) (setting this property automatically empties the contents of the list box).

Count

Returns the number of items in the list box (read-only)

LongInt = listbox.count

ItemIndex

Gets or sets the index (starting at 0) of the active selection in a single select box (undefined if the list box is a multiselect list box)

LongInt = listbox.ItemIndex

SelCount

Returns the number of items selected in a multiselect list box

LongInt = listbox.SelCount

Methods

addString

Adds a string to the list box

addString (str String)

deleteString

Deletes the string that resides at the specified index (starting at 0)

deleteString (index LongInt)

findString

Returns the index (starting at 0) of the string in the list box

LongInt findString (str String)

getText

Returns the string that resides at the specified index

string getText (index LongInt)

reset

Empties the list box (single or multi-select)

reset ( )

selRange

Highlights the specified range in a multi-select list box

selRange (lowIndex LongInt, highIndex LongInt)

getMultiSelAsCDL

Returns the indexes (in a comma-delimited string) of the items selected in a multi-select list box (This string can be fed to breakApart( ))

String getMultiSelAsCDL ( )

Events

onDblClick

Fires when the user double-clicks a selection in the list box. The active selection can be read using the getCurSel( ) method. If a user double-clicks the an immediate response is requested.

void onDoubleClick ( )

onSelChange

Fires when the user changes the active selection in the list box using keyboard moves (arrow keys) or mouse actions (single left click).

void onSelChange ( )

onKeyDown

Fires when the list box has focus and a key is pressed. Certain keys can be trapped using this event (e.g., ESCAPE and ENTER). The event passes in the virtual key code of the key that was pressed. This event fires before onSelChange( ) when arrow keys are used to change the selection in a list box. Because Windows generates this event on virtual key codes, it fires when you press the SHIFT, ALT, and CTRL keys as well as regular keys.

void onKeyDown (LongInt vkey)

{button ,AL(`NWC_INTRO;NWC_Valueprop;',0,"Defaultoverview",)} Related Topics

Progressbar

Properties

Values for the Progressbar properties can be specified with the Value property.

Pos

Gets or sets the position of the progressbar to an absolute value. Must be between the min and max value specified in setRangeAndStep ( ).

pos = LongInt (set)

LongInt = pos (get)

Methods

setRangeAndStep

Sets the minimum and maximum values for the progressbar and the step value (increment)

setRangeAndStep (minRange LongInt, maxRange LongInt, step LongInt)

stepIt

Steps the progressbar by the step value

stepIt ( )

Note

- The range and step value of the progressbar must be specified prior to use.

{button ,AL(`NWC_INTRO;NWC_Valueprop;',0,"Defaultoverview",)} Related Topics

Spinbox

Properties

Values for the Spinbox properties can be specified with the Value property.

min

Specifies the minimum (lower) value of the spinbox range (read-only)

max

Specifies the maximum (upper) value of the spinbox range (read-only)

Methods

Range

Sets the minimum and maximum values for the spinbox (If the range of a spinbox is not specified, the UIObject Value Property will return 0.)

range (minRange LongInt, maxRange LongInt)

Events

onChanged

Fired when the user changes the value in the spinbox (spinbox range must be specified prior to use).

Example

```
#spinbox3.range (LongInt (1), LongInt (10))
#spinbox3.value = LongInt (5)
message (“value = “, string (#spinbox3.value))
```

Note

- The Value property of a Spinbox native Windows control fails if you haven't defined its minimum and maximum range.

{button ,AL(`NWC_INTRO;NWC_Valueprop;',0,"Defaultoverview",)} Related Topics

TrackBar

Properties

Values for the TrackBar properties can be specified with the Value property.

Orientation

Determines whether the trackbar is vertical or horizontal (Set the property using the value of the enumerated constants).

tbHorizontal = 0

tbVertical = 2

TickMarks

Determines if the trackbar has tickmarks (Set the property using the value of the enumerated constants).

tsNoTicks = 0

tsAutoTicks = 1

TickStyle

Determines the style of the tickmarks on the trackbar (Set the property using the value of the enumerated constants).

tmBottomRight = 0

tmTopLeft = 4

tmBoth = 8

EnableSelRange

Determines whether the trackbar has a selectable range. If this is set to true, the properties SelStart and SelEnd are used to set the selected range; otherwise SelStart and SelEnd have no effect.

SelStart

Marks the beginning of the selected range if **EnableSetRange** is True

SelEnd

Marks the end of the selected range if **EnableSetRange** is False.

Min

Specifies the minimum value of the trackbar range

Max

Specifies the maximum value of the trackbar range

Pos

Specifies the position of the slider (thumb). Values must fall between the low and high range of the trackbar.

Style

Determines the style of the trackbar. The style property can be any of the TrackBarStyles Constants. You can set ***TickMarks***, ***TickStyle***, ***Orientation***, and ***EnableSelRange*** simultaneously using the **Style** property.

Methods

None

Events

endTrack

The user changed the track (thumb) position or clicked thumb without changing position. The endTrack event is always sent last and can be used as a generic onChanged event.

Sent when the user interacts with a trackbar using the mouse or the keyboard.

pageDown

The user pressed the PAGEDOWN key or clicked the channel below or to the right of the slider.

Sent when the user interacts with a trackbar by using the mouse or the keyboard

pageUp

The user pressed the PAGEUP key or clicked the channel above or to the left of the slider.

Sent when the user interacts with a trackbar using the mouse or the keyboard

moveBottom

The user pressed the END key.

Sent when the user interacts with a trackbar using the keyboard

lineDown

The user pressed the Right or Down arrow key.

Sent when the user interacts with a trackbar using the keyboard

lineUp

The user pressed the Left or Up arrow key.

Sent when the user interacts with a trackbar using the keyboard

moveTop

The user pressed the HOME key.

Sent when the user interacts with a trackbar using the keyboard

thumbPosition

The user released the Left mouse button following a ThumbTrack event.

Sent when the user interacts with a trackbar using the mouse

thumbTrack

The user dragged the slider with the mouse.

Sent when the user interacts with a trackbar using the mouse

Event

Reason sent

{button ,AL(`NWC_INTRO;NWC_Valueprop;',0,"Defaultoverview",)} **Related Topics**

AddinForm type

An add-in form is an external dynamic link library (DLL) that a third-party developer has provided. Not all DLLs can be used in Corel Paradox. To use a DLL in Corel Paradox, it must be designed so that it permits proper communication between it and Corel Paradox. For more information on a specific add-in and whether it can be used in Corel Paradox, contact the third-party developer who created it. For information on developing an add-in for Corel Paradox, refer to the Corel Paradox Developer Help for Corel Paradox Add-Ins.

If an add-in DLL has been designed for use in Corel Paradox, you can use the ObjectPAL AddinForm type methods to open and close the forms that the DLL contains and to obtain and set published form properties. An add-in DLL can also add menu options to the Corel Paradox menus.

Before an add-in form can be used in Corel Paradox, it must be registered.

Methods of the AddinForm type are similar to methods of the Form type.

Methods in the AddinForm type

attach
bringToTop
close
closeQuery
enumForms
getPosition
getPropertyAsInteger
getPropertyAsNumber
getPropertyAsString
getTitle
hide
isAssigned
isMaximized
isMinimized
isVisible
maximize
menuAction
minimize
open
postMessage
sendMessage
setPosition
setProperty
setTitle
show
wait
windowHandle

Print related ObjectPAL methods and examples

attach method

Associates an AddinForm variable with an open add-in form.

Syntax

```
1. attach ( const formTitle String ) Logical
2. attach ( const windowHandle Longint ) Logical
```

Description

attach associates an AddinForm variable with an open add-in form. This method returns True if it succeeds; otherwise, it returns False.

Syntax 1 associates the AddinForm variable to the form whose title is indicated by *formTitle*. The argument *formTitle* specifies a form's title as displayed in the Title Bar, not the add-in form's name. You can obtain an add-in form's title by calling the ObjectPAL method **getTitle**.

Syntax 2 attaches the AddinForm variable to the add-in form window indicated by *windowHandle*.

Note

- **Attach** can successfully associate the AddinForm variable regardless of whether the add-in form is hidden or visible.

Example

{button ,AL(`OPAL_TYPE_ADDINFORM;OPAL_METH_AFOPEN;OPAL_METH_AFSHOW;OPAL_METH_AFSETTITLE;OPAL_METH_AFWINDOWHANDLE;',0,"Defaultoverview",)} Related Topics

attach example

The following example shows code which should be attached to the **pushButton** method of a form. A String value is used to prompt the user for the form title and then the AddinForm variable is attached to the specified form. Assume that the add-in form window is already open on the Corel Paradox desktop.

```
; AttachToAddin::pushButton
method pushButton(var eventInfo Event)
var
    AF         AddinForm
    formTitle  String
endVar

Try
   formTitle.view("Enter add-in form window title")
   AF.Attach(formTitle)
   msgInfo("Attach...", "succeeded")
onFail
   msgInfo("Attach failed", formTitle+" add-in form not found")
   errorClear()
endTry
endMethod
```

bringToTop method

Brings the add-in form window to the forefront and makes it active.

Syntax

```
bringToTop ( )
```

Description

When several windows are displayed, they seem to overlap, giving an appearance of layers. Use **bringToTop** to display an add-in form's window on the top of the stack. **bringToTop** makes the add-in form the active window.

If a **hide** statement has made an add-in form invisible, **bringToTop** makes it visible again.

Example

{button ,AL(`OPAL_TYPE_ADDINFORM;OPAL_METH_AFISVISIBLE;OPAL_METH_AFHIDE;OPAL_METH_AFSHOW;',0,"Defaultoverview",)} Related Topics

bringToTop example

In this example, the **pushButton** method for a button named *ShowItemsList* makes the *Items* add-in form window the topmost layer. If the *Items* add-in form window is hidden, it is automatically made visible when it is brought to the top. Assume that *Items* is open on the desktop and has the window title ItemList.

```
; ShowItemsList::pushButton
method pushButton(var eventInfo Event)
var
   AF        AddinForm
endVar
AF.attach("ItemList")   ; attaches to Items by title
AF.bringToTop()         ; makes the Items window the topmost layer
                        ; and unhides it (if hidden)
endMethod
```

close method

Closes an add-in form window.

Syntax

close ( ) Logical

Description

close closes an add-in form window.

Example

{button ,AL(`OPAL_TYPE_ADDINFORM;OPAL_METH_AFOPEN;OPAL_METH_AFATTACH;',0,"Defaultoverview",)} Related Topics

close example

In this example, the AddinForm variable *AF* is declared in the Var window at the form level. The form has various pushButtons that open the add-in form, obtain specific properties, change specific properties, and manipulate the add-in form window. The following code appears on the pushbutton labeled Close, and when selected, closes the add-in form.

```
;Close::pushButton
method pushButton(var eventInfo Event)
    AF.Close()
endMethod
```

closeQuery method

Closes a query.

Syntax

```
closeQuery ( ) Logical
```

Description

closeQuery asks (queries) an add-in form whether it is willing to close.

This method returns True if the add-in form is willing to close and False if the add-in form is not willing to close.

Example

{button ,AL(`OPAL_TYPE_ADDINFORM;OPAL_METH_AFCLOSE;OPAL_METH_AFHIDE;',0,"Defaultoverview",)} Related Topics

closeQuery example

In this example, the custom procedure **CloseSubForm** is defined in the Proc window at the form level. The procedure attempts to close the add-in form after asking the add-in form if it is willing to be closed. If **closeQuery** returns True, the add-in form is closed. If **closeQuery** returns False, a message is displayed to the user indicating a course of action to follow that corrects the situation. Assume the AddinForm variable *AF* is defined in the Var window at the form level and that the add-in form is already open and assigned.

```
; #Form1::proc
Proc CloseSubForm() Logical
   if AF.closeQuery() then       ; add-in form is willing to close
      AF.close()
      return True
   else                          ; add-in form is not willing to close
      msgStop("error","Please fill out the entire form")
      AF.bringToTop()
      Return False
   endIf
endProc
```

enumForms method

Creates an array listing registered add-in forms.

Syntax

```
enumForms ( var formNames Array[ ] String ) Logical
```

Description

enumForms fills the array *formNames* with a list of add-in forms that have been registered in Corel Paradox. You declare *formNames* as a resizeable array before calling this method.

An add-in form must be registered before it can be used in Corel Paradox.

Example

{button ,AL(`OPAL_TYPE_ADDINFORM;OPAL_METH_AFOPEN;idh_pals_edregaddin;',0,"Defaultoverview",)} **Related Topics**

enumForms Examples

Example 1 Listing the names of add-in forms to an array

Example 2 Using **enumForms** to populate a list object on a form, with the names of add-in forms from Corel Paradox

enumForms example 1

This example lists the names of add-in forms in the array *openForms* and then displays openForms.

```
; getForms::pushButton
method pushButton(var eventInfo Event)
var
  AF          AddinForm
  availForms  Array[] String
endVar

AF.enumForms(availForms)
availForms.view()            ; Lists names of registered add-in forms.
endMethod
```

enumForms example 2

This example uses **enumForms** to populate a list object on a form with the names of add-in forms registered in Corel Paradox. enumForms generates the array *ArList*, which is used as input for the list field. Assume that the list field is named FormList and the list object within FormList is called List.

```
method pushButton(var eventInfo Event)
var
   AF       AddinForm
   ArList   Array [] String
   i        LongInt
endVar
AF.enumForms( ArList )
FormList.List.List.Count = ArList.size()
For i from 1 to ArList.size()
  FormList.List.List.Selection = i
  FormList.List.List.Value = ArList[i]
EndFor
endMethod
```

getPosition method

Returns the position (in twips) of an add-in window.

Syntax

```
getPosition ( var x LongInt, var y LongInt, var w LongInt, var h LongInt )
```

Description

getPosition finds the position of the add-in form's window. The arguments *x* and *y* contain the horizontal and vertical coordinates of the upper-left corner of the form (in twips), and *w* and *h* contain the width and height (in twips) of the form.

To ObjectPAL, the screen is a two-dimensional grid with the origin (0, 0) at the upper-left corner of an object's container, positive x values extending to the right, and positive y values extending down.

For dialog boxes and pop-up forms, the position is given relative to the entire screen. For multiple document interface (MDI) child forms (default), the position is given relative to the Corel Paradox desktop.

Example

{button ,AL(`OPAL_TYPE_ADDINFORM;OPAL_METH_AFSETPOSITION;OPAL_METH_AFSETPROPERTY;OPAL_METH_AFGETPROPERTYASNUMBER;OPAL_METH_AFGETPROPERTYASINTEGER OPAL_METH_AFGETPROPERTYASSTRING;',0,"Defaultoverview",)} Related Topics

getPosition example

The following example shows code which appears in the **pushButton** method on a form. The coordinates of the add-in form window are displayed by calling the method **msgInfo**.

```
method pushButton(var eventInfo Event)
var
     AF         AddinForm
     x,y,h,w    LongInt
endVar

     AF.open( "add-in" )   ; where add-in is the add-in form
     AF.show()
     AF.getPosition( x, y, w, h )
     msgInfo("Add-in position:",string( x, ",", y, "   ", w ," x ", h))
endMethod
```

getPropertyAsInteger method

Returns the value of the specified property of an add-in form as a LongInt.

Syntax

```
getPropertyAsInteger ( const propertyName String ) LongInt
```

Description

getPropertyAsInteger returns the value of the add-in's property specified by *propertyName*. The value of the property is returned as a long integer.

Use **getPropertyAsNumber** for properties that return a floating-point number, and **getPropertyAsString** for properties that return a String value.

Note

- The add-in form is responsible for performing type conversions as necessary to return a LongInt value.

Example

{button ,AL(`OPAL_TYPE_ADDINFORM;OPAL_METH_AFSETPROPERTY;OPAL_METH_AFSETTITLE;',0,"Defaultoverview",)} Related Topics

getPropertyAsInteger example

In this example, the user is prompted to enter the name of a property of the add-in form which returns an Integer value. The current value of the specified property is displayed in a **msgInfo** window if the method is successful.

```
method pushButton(var eventInfo Event)
var
   AF         AddinForm
   propName   String
endVar

try
   AF.open( "add-in" )   ; where add-in is the name of the
                         ; add-in form
   AF.show()
   propName.view("Property that returns an integer:")
   msgInfo(propName+" is:",AF.getPropertyAsInteger(propName))
onFail
   msgInfo("getPropertyAsInteger("+propName+") failed.","Property
            unknown or could not return an integer value")
   errorClear()
endTry
endMethod
```

getPropertyAsNumber method

Returns the value of the specified property of an add-in form as a floating-point number.

Syntax

```
getPropertyAsNumber ( const propertyName String ) Number
```

Description

getPropertyAsNumber returns the value of the add-in's property specified by *propertyName*. The value of the property is returned as a number.

Use **getPropertyAsInteger** for properties that return an integer, and **getPropertyAsString** for properties that return a String value.

Note

- The add-in form should perform type conversions as necessary to return a floating-point Number value.

Example

{button ,AL(`OPAL_TYPE_ADDINFORM;OPAL_METH_AFSETPROPERTY;OPAL_METH_AFSETTITLE;',0,"Defaultoverview",)} Related Topics

getPropertyAsNumber example

In this example, the user is prompted to enter the name of a property of the add-in form that returns a floating-point numeric value. The current value of the specified property is displayed in a **msgInfo** window if the method is successful.

```
method pushButton(var eventInfo Event)
var
     AF         AddinForm
     propName   String
endVar
try
   AF.open( "add-in" )   ; where add-in is the name of the
                         ; add-in form
   AF.show()
   propName.view("Property that returns a number:")
   msgInfo(propName+" is:",AF.getPropertyAsNumber(propName))
onFail
   msgInfo("getPropertyAsNumber("+propName+") failed.","Property
            unknown or could not return a float number")
   errorClear()
endTry
endMethod
```

getPropertyAsString method

Returns the value of the specified property of an add-in form as a String.

Syntax

```
getPropertyAsString ( const propertyName String ) String
```

Description

getPropertyAsString returns the value of the add-in's property specified by *propertyName*. The value of the property is returned as a String value.

Use **getPropertyAsInteger** for properties that return an integer, and **getPropertyAsNumber** for properties that return a floating-point number.

Note

- The add-in form should perform type conversions as necessary in order to return a String value.

Example

{button ,AL(`OPAL_TYPE_ADDINFORM;OPAL_METH_AFSETPROPERTY;OPAL_METH_AFSETTITLE;',0,"Defaultoverview",)} Related Topics

getPropertyAsString example

In this example, the user is prompted to enter the name of a property of the add-in form that returns a String value. The current value of the specified property is displayed in a **msgInfo** window if the method is successful.

```
method pushButton(var eventInfo Event)
var
     AF         AddinForm
     propName   String
endVar
try
   AF.open( "add-in" )   ; where add-in is the name of the
                         ; add-in form
   AF.show()
   propName.view("Property that returns a string:")
   msgInfo(propName+" is:",AF.getPropertyAsString(propName))
onFail
   msgInfo("getPropertyAsString("+propName+") failed.","Property
            unknown or could not return a String")
   errorClear()
endTry
endMethod
```

getTitle method

Returns the text on the window Title Bar of the add-in form's window.

Syntax

```
getTitle ( ) String
```

Description

getTitle returns the text on the Title Bar of the add-in form's window.

Example

{button ,AL(`OPAL_TYPE_ADDINFORM;OPAL_METH_AFSETTITLE;OPAL_METH_AFWINDOWHANDLE;OPAL_METH_AFATTACH;OPAL_METH_AFOPEN;',0,"Defaultoverview",)} Related Topics

getTitle example

In this example, the **pushButton** method for the *showTitle* button opens a Calculator add-in form and returns the add-in's title. A dialog box opens where you can make changes to the title. It then checks to see if the title has been modified by the user and if so, calls the **setTitle** method to change the add-in's window title as specified.

```
; showTitle::pushButton
method pushButton(var eventInfo Event)
var
   AF         AddinForm
   origTitle,
   newTitle   String
endVar

AF.open("Calculator")          ; open add-in form but do not
                               ; make it visible
origTitle=af.getTitle()
newTitle=origTitle             ; comparison value
newTitle.view("Enter new title for add-in")
if origTitle<>newTitle then
   AF.setTitle(newTitle)       ; call setTitle only if changes made
endIf
AF.show()                      ; display the add-in with its new title
endMethod
```

hide method

Makes an add-in form window invisible.

Syntax

```
hide ( )
```

Description

hide makes an add-in form window invisible but doesn't close it.

Example

{button ,AL(`OPAL_TYPE_ADDINFORM;OPAL_METH_AFSHOW;OPAL_METH_AFOPEN;OPAL_METH_AFISVISIBLE;',0,"Defaultoverview",)} Related Topics

hide example

In this example, the **pushButton** method for the *hideForm* button attaches to an add-in form which is open on the Corel Paradox desktop. The method hides the add-in form and then shows it again. Assume that the open add-in form is named Calculator.

```
; hideForm::pushButton
method pushButton(var eventInfo Event)
var
   AF     AddinForm
endVar
AF.attach("Calculator")                 ; attaches to open add-in form
AF.hide()                               ; makes form invisible
msgInfo("Status:","Add-in form has been hidden")
AF.show()                               ; make form visible again
if AF.isVisible() then
   msgInfo("Status", "It's visible.")
endIf
endMethod
```

isAssigned method

Returns whether an AddinForm variable has been assigned a value.

Syntax

```
isAssigned ( ) Logical
```

Description

isAssigned returns True if the AddinForm variable has been assigned a value; otherwise, it returns False.

Example

{button ,AL(`OPAL_TYPE_ADDINFORM;OPAL_METH_AFOpen;OPAL_METH_AFCLOSE;OPAL_METH_AFATTACH;',0,"Defaultoverview",)} **Related Topics**

isAssigned example

In this example, **isAssigned** is used to determine if an add-in form is open before attempting to set the add-in form's *QueryCustId* property. If **isAssigned** returns False, a message is displayed onscreen and the code that manipulates the add-in form is not executed. Assume that the variable *AF* is declared in the Var window at the form level and that the *QueryCustId* property of the add-in form requires a numeric value representing the Customer ID to use for the query.

```
method pushButton(var eventInfo Event)
var
   custId   Number
endVar
if NOT AF.isAssigned() then
   ; querying of the customer information can't happen because the
   ; customer information application isn't running
   msgInfo("Sorry","You must first open the QueryCust application")
else
   ; The following code asks the user for a customer ID number and sets
   ; the QueryCustId property of the add-in form to that value.
   ; Assume the add-in form automatically recalculates the displayed
   ; information when the QueryCustId value is changed.
   custId.view("Enter customer ID to view")
   AF.setProperty("QueryCustId",custId)
endIf
endMethod
```

isMaximized method

Reports whether an add-in form window is displayed at its maximum size.

Syntax

```
isMaximized ( ) Logical
```

Description

isMaximized returns True if an add-in form is displayed full screen; otherwise, it returns False.

Example

{button ,AL(`OPAL_TYPE_ADDINFORM;OPAL_METH_AFMAXIMIZE;OPAL_METH_AFMINIMIZE;OPAL_METH_AFISMINIMIZED;',0,"Defaultoverview",)} Related Topics

isMaximized example

The following example shows code which is placed in the custom method **displayAsOpposite** at the form level. It attaches to an open add-in form, tests the add-in's current display state and then changes the form's state to it's opposite setting. For example, if the add-in form is maximized, its state is changed to minimized. Assume that the add-in form has the window title Postage App and is already open on the Corel Paradox desktop. The AddinForm variable AF is declared in the Var window at the form level.

```
method displayAsOpposite()
AF.attach("Postage App")
if AF.isMaximized() then
   AF.minimize()             ; displayed maximized so minimize it.
else
   if AF.isMinimized() then
      AF.maximize()          ; displayed minimized so maximize it
   else
      if AF.isVisible() then
         AF.hide()           ; displayed in a normal window so hide it
      else
         AF.show()           ; it's hidden so show it
      endIf
   endIf
endIf
endMethod
```

isMinimized method

Reports whether a window is displayed as an icon.

Syntax

isMinimized ( ) Logical

Description

isMinimized returns True if a form is displayed as an icon; otherwise, it returns False.

Example

{button ,AL(`OPAL_TYPE_ADDINFORM;OPAL_METH_AFISMAXIMIZED;OPAL_METH_AFMINIMIZE;OPAL_METH_AFISMAXIMIZED;OPAL_METH_AFSHOW;OPAL_METH_AFHIDE;',0,"Defaultoverview",)} Related Topics

isMinimized example

This example attaches to an open add-in form, tests its current state, and then changes the state of the form. If the add-in form is displayed minimized, the application is set to display as a window. If the add-in form is displayed as a window, it is minimized to display as an icon. If the add-in form is hidden, this method does not alter its display. Assume that the add-in form has the window title Calculator and is already open on the Corel Paradox desktop.

```
method pushButton(var eventInfo Event)
var
     AF     AddinForm
endVar

AF.attach("Calculator")
if AF.isVisible() then
   if AF.isMinimized() then
      AF.show()
   else
      AF.minimize()
      msgInfo("Is window minimized?",AF.isMinimized())
   endIf
else
   msgInfo("Status:","Add-in is hidden")
endIf
endMethod
```

isVisible method

Reports whether an add-in window is displayed.

Syntax

isVisible ( ) Logical

Description

isVisible returns True if any part of a window is displayed (not hidden); otherwise, it returns False.

Example

{button ,AL(`OPAL_TYPE_ADDINFORM;OPAL_METH_AFHIDE;OPAL_METH_AFSHOW;',0,"Defaultoverview",)} **Related Topics**

isVisible example

See the example for **isMaximized**.

maximize method

Maximizes an add-in form's window.

Syntax

```
maximize ( )
```

Description

maximize displays an add-in window at its full size. Calling this method is equivalent to choosing Maximize from the Control menu.

Example

{button ,AL(`OPAL_TYPE_ADDINFORM;OPAL_METH_AFMINIMIZE;OPAL_METH_AFISMINIMIZED;OPAL_METH_AFISMAXIMIZED;OPAL_METH_AFSHOW;',0,"Defaultoverview",)} Related Topics

maximize example

See the example for **isMaximized**.

menuAction method

Sends an event to an add-in form's **menuAction** event handler.

Syntax

```
menuAction ( const menuID LongInt ) Logical
```

Description

menuAction constructs a MenuEvent and calls the add-in form's **menuAction** event handler. The action taken as a result of this method is determined solely by the developer of the add-in form. There is no default behavior.

Because the **menuAction** event is handled by the add-in form and not by Corel Paradox, use of ObjectPAL's MenuCommands constants may generate unexpected results.

Example

{button ,AL(`OPAL_TYPE_ADDINFORM;OPAL_METH_AFSENDMESSAGE;',0,"Defaultoverview",)} Related Topics

menuAction example

In this example, the menu ID value of 17 is sent to the menu event handler for the add-in form *AF.* Assume that the menu ID17 causes the add-in form to execute the custom code stored within it.

```
method pushButton(var eventInfo Event)
var
  AF     AddinForm
endVar

AF.open("My Form")
AF.menuAction(17)
endMethod
```

minimize method

Minimizes an add-in form's window.

Syntax

```
minimize ( )
```

Description

minimize displays an add-in form's window as an icon. Calling this method is equivalent to choosing Minimize from the Control menu.

Example

{button ,AL(`OPAL_TYPE_ADDINFORM;OPAL_METH_AFMAXIMIZE;OPAL_METH_AFISMINIMIZED;OPAL_METH_AFISMINIMIZED;OPAL_METH_AFSHOW;',0,"Defaultoverview",)} **Related Topics**

minimize example

See the example for **isMaximized**.

open method

Opens an add-in form window.

Syntax

```
open (const formName String [ , const dialog Logical, const visible Logical ] )
```

Description

open displays the add-in form specified in *formName*. The optional argument *dialog* specifies whether the add-in form should be opened as a dialog box. The optional argument *visible* specifies whether the form should be opened and visible.

An add-in form cannot be opened in design window. You must design a form in the application in which it was created.

An add-in form must be registered before it can be used in Corel Paradox.

Example

{button ,AL(`OPAL_TYPE_ADDINFORM;OPAL_METH_AFCLOSE;OPAL_METH_AFSHOW;OPAL_METH_AFATTACH;idh_pals_edregaddin;',0,"Defaultoverview",)} Related Topics

open example

In this example, the **pushButton** method of a form contains code that attaches an AddinForm variable to an add-in form. If the add-in form is open on the Corel Paradox desktop, the method attaches to the open window. If the add-in is not already open, the method opens it. Once the AddinForm variable is assigned, the add-in form is made visible. Assume that the add-in form is called Calculator and has been registered in Corel Paradox.

```
method pushButton(var eventInfo Event)
var
   AF    AddinForm
endVar

; attach to open Calculator window if there is one
if NOT AF.attach("Corel Paradox Calculator") then ;use add-in window title
    AF.open("Calculator")                  ;use registered name
endIf
if AF.isAssigned() then
    AF.show()                      ; display the add-in
else
    msgInfo("Problem(?)","AddinForm variable not assigned")
endIf
endMethod
```

postMessage method

Posts a message to Windows.

Syntax

```
postMessage ( const command LongInt, const wParam LongInt, const lParam LongInt )
```

Description

postMessage posts a message to Windows. Unlike **sendMessage**, which dispatches its message immediately, this method adds its message to the end of the Windows message queue and dispatches it after the messages (if any) ahead of it.

Valid arguments to this method are determined by Windows, not by Corel Paradox. For more information, see your Windows programming documentation.

Note

- **postMessage** should only be used by Windows programmers.

Example

{button ,AL(`OPAL_TYPE_ADDINFORM;OPAL_METH_AFSENDMESSAGE;OPAL_METH_SYWINGETMESSAGEID;OPAL_METH_SYWINSENDMESSAGE;OPAL_METH_MNDATA;OPAL_METH_MNID;OPAL_METH_FOWINDOWCLIENTHANDLE;OPAL_METH_FOWINDOWHANDLE;',0,"Defaultoverview",)} Related Topics

postMessage example

See the example for **sendMessage**.

sendMessage method

Sends a message to Windows.

Syntax

```
sendMessage ( const command LongInt, const wParam LongInt, const lParam LongInt ) LongInt
```

Description

sendMessage sends a message to Windows. Valid arguments to this method are determined by Windows, not by Corel Paradox. For more information, see your Windows programming documentation.

Note

- **sendMessage** should only be used by Windows programmers.

Example

{button ,AL(`OPAL_TYPE_ADDINFORM;OPAL_METH_AFPOSTMESSAGE;OPAL_METH_SYWINGETMESSAGEID;OPAL_METH_SYWINPOSTMESSAGE;OPAL_METH_MNDATA;OPAL_METH_MNID;OPAL_METH_FOWINDOWCLIENTHANDLE;OPAL_METH_FOWINDOWHANDLE;',0,"Defaultoverview",)} Related Topics

sendMessage example

In this example, two messages are defined in the constants section of a script window and are sent with different results. The add-in form processes the message WM_UserMSG1 immediately. However, the message WM_UserMSG2 isn't processed until there is a pause for message processing. (The pause can occur as a result of an explicit call to SLEEP() or any other action that causes a wait). The effect is that WM_UserMSG1 will be processed before WM_UserMSG2.

Note

- The author of the add-in form must provide code on the add-in side to handle these messages because they are not standard Windows messages.

```
method run(var eventInfo Event)
const
  WM_UserMSG1 = WM_USER+1
  WM_UserMSG2 = WM_USER+2
endConst
var
  AF       AddinForm
  result   LongInt
endVar

  AF.open("Test Form")
  AF.postMessage( WM_UserMSG2, 0, 0 )
  result = AF.sendMessage( WM_UserMSG1, 0, 0 )
endMethod
```

setPosition method

Positions an add-in window on screen.

Syntax

```
setPosition ( const x LongInt, const y LongInt, const w LongInt, const h LongInt )
```

Description

setPosition positions an add-in form's window onscreen. The arguments *x* and *y* specify the coordinates of the upper-left corner of the form (in twips), and *w* and *h* specify the width and height (in twips) of the form.

To ObjectPAL, the screen is a two-dimensional grid with the origin (0, 0) at the upper-left corner of an object's container, positive x values extending to the right, and positive y values extending down.

For dialog boxes and pop-up forms, the position is given relative to the entire screen. For multiple document interface (MDI) child forms (default), the position is given relative to the Corel Paradox desktop.

Example

{button ,AL(`OPAL_TYPE_ADDINFORM;OPAL_METH_AFGETPOSITION;OPAL_METH_AFSETPROPERTY;OPAL_METH_AFGETPROPERTYASNUMBER;OPAL_METH_AFGETPROPERTYASINTEGER;OPAL_METH_AFGETPROPERTYASSTRING;OPAL_METH_AFOPEN;;',0,"Defaultoverview",)} **Related Topics**

setPosition example

In this example, the add-in form Calculator is opened and its current position displayed on the message line. The method then prompts the user to enter new position coordinates for the Calculator.

```
method run(var eventInfo Event)
var
   AF            AddinForm
   choice,
   position      String
   x,y,w,h       LongInt
endVar

AF.open("Calculator")
AF.show()
AF.getposition(x,y,w,h)
position= string(x)+", "+string(y)+"    "+string(w)+" x "+string(h)
message("Current position: "+position)
choice=msgQuestion("Do you want to move the add-in to a new
       position","")
if choice="Yes" then
   x.view("Enter upper-left X coordinate")
   y.view("Enter upper-left Y coordinate")
   w.view("Enter width of window")
   h.view("Enter height of window")
   AF.setPosition( X, Y, W, H )
endIf
endMethod
```

setProperty method

Lets you change the value of a property of an add-in form.

Syntax

```
setProperty ( const propertyName String, const propertyValue AnyType )
```

Description

setProperty lets you change the value of a property of an add-in form. The property to change is specified in *propertyName* and can be any property recognized by the add-in form. The value of the property is specified in *propertyValue*.

Based on the data type of *propertyValue*, Corel Paradox calls the add-in form's appropriate setProperty method as String, Integer, or floating-point number.

Example

{button ,AL(`OPAL_TYPE_ADDINFORM;OPAL_METH_AFSETPROPERTY;OPAL_METH_AFGETPROPERTYASNUMBER;OPAL_METH_AFGETPROPERTYASINTEGER;OPAL_METH_AFGETPROPERTYASSTRING;OPAL_METH_SETTITLE;OPAL_METH_GETTITLE;',0,"Defaultoverview",)} Related Topics

setProperty example

See example for **isAssigned**.

setTitle method

Sets the text on the Title Bar.

Syntax

```
setTitle ( const text String )
```

Description

setTitle changes the text on the Title Bar to the text specified in *text*. The maximum length of *text* is 78 characters. If you change a form's title, remember that you must use the new title when you want to attach to that form. For more information, see the description of **attach**.

Example

{button ,AL(`OPAL_TYPE_ADDINFORM;OPAL_METH_GETTITLE;OPAL_METH_AFSETPROPERTY;OPAL_METH_AFGETPROPERTYASNUMBER;OPAL_METH_AFGETPROPERTYASINTEGER;OPAL_METH_AFGETPROPERTYASSTRING;',0,"Defaultoverview",)} Related Topics

setTitle example

See the **getTitle** example.

show method

Displays a minimized window at its previous size. Makes a hidden form visible.

Syntax

```
show ( )
```

Description

show restores a minimized window to the size it was before it was minimized. **show** also makes a hidden form visible. This method is similar to the Restore command on the Control menu.

show doesn't make an add-in form the topmost window; use bringToTop to make an add-in form the top layer.

Example

{button ,AL(`OPAL_TYPE_ADDINFORM;OPAL_METH_AFHIDE;OPAL_METH_AFISVISIBLE;',0,"Defaultoverview",)} **Related Topics**

show example

See the **getTitle** example.

wait method

Suspends execution of a method.

Syntax

```
wait ( ) AnyType
```

Description

wait suspends execution of the current method until the add-in form you're waiting for closes. This method is useful when you open an add-in form as a dialog box. Execution resumes in the calling form when the add-in form you're waiting for (the called form) closes.

The return value for **wait** depends on what you specify when creating the add-in form. Typically, 1 is returned when you close the add-in form by clicking OK, 2 is returned when you close the add-in form by clicking Cancel.

Example

{button ,AL(`OPAL_TYPE_ADDINFORM;OPAL_METH_AFOPEN;OPAL_METH_AFHIDE;OPAL_METH_AFSHOW;',0,"Defaultoverview",)} Related Topics

wait example

In this example, the add-in form is opened from a **pushButton** method on a form. A wait is called and the return value is tested. If the return value is 1, a **msgInfo** dialog box is displayed and the add-in form is closed.

```
method pushButton(var eventInfo Event)
var
  AF    AddinForm
endVar
AF.open("Data")
if AF.wait()=1 then
   msgInfo("Data Entered",AF.getPropertyAsString("User Data"))
endIf
AF.close()
endMethod
```

windowHandle method

Returns the handle of an add-in form's window.

Syntax

windowHandle ( ) LongInt

Description

A window handle is a unique integer identifier assigned to a window by Windows. **windowHandle** returns an integer value representing the window handle of an add-in form.

This method should be used only by advanced programmers.

This information is useful only if you're using functions from a dynamic link library (DLL).

Example

{button ,AL(`OPAL_TYPE_ADDINFORM;OPAL_METH_AFATTACH;OPAL_METH_TCATTACH;',0,"Defaultoverview",)} Related Topics

windowHandle example

In this example, the **pushButton** method of a form's *WndwHndle* button calls an external routine *EnableWindow. EnableWindow* is defined in the Uses window at the form level and takes the window handle of the add-in form as its parameter.

The following code appears in the Uses window at the form level:

```
uses USER32
  EnableWindow( hWnd CLONG, bEnable CLONG) CLONG
endUses
```

The following code appears in the WndwHndle **pushButton** method on the form:

```
; WndwHndle::pushButton

method pushButton(var eventInfo Event)
var
  AF    AddinForm
endVar
  AF.open("myForm")                   ; open the add-in form
  EnableWindow( AF.windowHandle(), 1) ; call the external routine and
                                      ; pass the add-in's window handle
  AF.close()
endMethod
```

Configuration settings in the system registry

Corel Paradox reads the system registry for settings that configure the session. Most settings that are supported in .INI files can also be set in the system registry.

Use the registry editor supplied with Windows to view or modify the system registry file. In Windows 95, the registry editor is REGEDIT.EXE; in Windows NT, it is REGEDT32.EXE. Check your Windows documentation for information on how to use these editors.

The registry keys which are specific to Corel Paradox are found at

- HKEY_CURENT_USER\SOFTWARE\COREL\PARADOX\8.0

In addition, the following registry keys are required for the Borland Database Engine (BDE):

- HKEY_LOCAL_MACHINE\SOFTWARE\BORLAND\BLW32
- HKEY_LOCAL_MACHINE\SOFTWARE\BORLAND\DATABASE ENGINE

Note

- Only advanced developers should modify of the system registry. Changes you make using the registry editor can't be undone.

{button ,AL(`commandline;iniFiles;systemRegistryWININI;',0,"Defaultoverview",)} **Related Topics**

WIN.INI settings in the system registry

Earlier versions of Corel Paradox stored configuration settings in the [PDOXWIN] and [IDAPI] sections of WIN.INI. These settings now reside in the system registry. Unless otherwise stated, the system registry is the only location where these values can be set.

UserName

The user name is specified by the value of the UserName string at HKEY_CURRENT_USER\SOFTWARE\COREL\PARADOX\8.0\CONFIGURATION KEY.

Company

The company name is specified by the value of the Company string at the HKEY_CURRENT_USER\SOFTWARE\COREL\PARADOX\8.0\CONFIGURATION KEY.

HomeDir

The home directory location is specified by the value of the <Default> string at HKEY_CURRENT_USER\SOFTWARE\COREL\PARADOX\8.0\HOMEDIR.

PrivDir

The private directory location is specified by the value of the <Default> key at HKEY_CURRENT_USER\SOFTWARE\COREL\PARADOX\8.0\PRIVDIR.

The private directory may be specified by the **-p** command-line option.

WorkDir

The working directory location is specified by the value of the <Default> key at HKEY_CURRENT_USER\SOFTWARE\COREL\PARADOX\8.0\WORKDIR.

The working directory may be specified by the **-w** command-line option.

CONFIGFILE01

The location (full path name) and, optionally, the name of the Borland Database Engine (BDE) configuration file is specified by the value of the CONFIGFILE01 string located at HKEY_LOCAL_MACHINE\SOFTWARE\BORLAND\DATABASE ENGINE.

The BDE configuration file can be specified using the -**o** command-line option.

DLLPATH

The location where the BDE files can be found is specified by the value of the DLLPATH string located at HKEY_LOCAL_MACHINE\SOFTWARE\BORLAND\DATABASE ENGINE.

Note

- For more information about the -**p**,-**w**, and -**o** command line options, see Command-line options.

{button ,AL(`commandline;iniFiles;systemRegistry;',0,"Defaultoverview",)} Related Topics

Using .INI Files

Corel Paradox reads .INI files for settings that override the system registry settings. Corel Paradox uses the following .INI files:

- PDOXWIN.INI file. You can call PDOXWIN.INI by using the **-i** command-line option.
- PDOXWORK.INI, located in working directories. You can call PDOXWORK.INI or an alternate file using the **-d** command-line option.

To customize menus, you enter keys and values in the system registry. For instructions, see PXDLITE.INI settings in the system registry.

{button ,AL(`commandline;systemRegistry;systemRegistryWININI;',0,"Defaultoverview",)} Related Topics

Using command-line options

To start Corel Paradox with one or more command-line options

1. Do one of the following:

- In Windows 95, choose Run from the Windows Start menu.
- In Windows NT, click File, Run in the Program Manager.

2. Type `pdxwin32`, and add the option(s) you want to use.

If you use more than one option, separate each with a space.

For a list of available command-line options, see Command-line options.

To start Corel Paradox with the same command-line options

Do one of the following:

- In Windows 95, right-click and use the Windows Properties dialog box to change the properties of the Corel Paradox 8 icon.
- In Windows 95, create a new link containing the command-line options in its Shortcut settings.
- Use the registry editor to add the FLAGS string name to your system registry. The FLAGS string name is found at HKEY_CURRENT_USER\SOFTWARE\COREL\PARADOX\8.0\CONFIGURATION. You may need to create the FLAGS string name if you do not already have an entry at this location. Check your Windows documentation for information about the registry editor.

{button ,AL(`systemRegistry;iniFiles;toUseCommandlineOptions;systemRegistryWININI;idh_t_config_pdoxworkini;idh_t_config_pfwwinini;',0,"Defaultoverview",)} Related Topics

PXDLITE.INI settings in the system registry

PDXLITE.INI contains developer menu settings which are now stored in the system registry at HKEY_CURRENT_USER\SOFTWARE\COREL\PARADOX\8.0\PXDLITE.

You can add menu commands to the File, Tools, and Help menus and the File, Open submenu using the corresponding keys within PXDLITE. Inside a key, you place values corresponding to the menu command you want to add.

The strings placed in PXDLITE are specified using the following syntax:

String name	Data value
MenuChoice	drive:\path\filename [function] where [function] is the name of a function within a .DLL.

Adding an ampersand before a letter in the menu command makes that menu command identifiable by that letter.

The filename can be any Corel Paradox form (.FSL, .FDL), script (.SSL, .SDL), or program file (.EXE, .COM, .BAT, and .PIF), or .DLL. All other filenames are ignored.

The key for the File menu is File, the Tools menu is Tools, and the Help menu is Help. The key for the File, New submenu is New and the key for the File, Open submenu is Open.

Example

To add the Paint Brush command to the Tools menu, with accelerator key B:

1. Using the registry editor, add PXDLITE key under HKEY_CURRENT_USER\SOFTWARE\COREL\PARADOX\8.0.
2. Under PXDLITE, add the new key Tools.
3. Under Tools, add a new string value, Paint&Brush, with the value "C:\WINDOWS\PBRUSH.EXE".

To specify a particular order for your custom menu options, precede the menu's value with an integer value (starting with 0), followed by a comma.

When you close the registry editor and start Corel Paradox, the new command appears on the Tools menu and runs Windows Paintbrush when you click it or press ALT + TAB.

Note

- Only advanced developers should modify of the system registry. Changes you make using the registry editor can't be undone.

{button ,AL(`commandline;iniFiles;systemRegistry;systemRegistryWININI;',0,"Defaultoverview",)}
Related Topics

PDOXWORK.INI

PDOXWORK.INI describes additional entries to the working directory's Viewer. Corel Paradox creates this file in the directory you set as your working directory. The [Folder] section lists the name of the file and the position of the icon for each reference you have created.

You can use PDOXWORK.INI to override settings in PDOXWIN.INI and the system registry. To do so, start Corel Paradox with the command-line setting **-d PDOXWORK.INI**.

{button ,AL(`commandline;inifiles;systemRegistry;systemRegistryWININI;idh_t_config_pxdliteini;',0, "Defaultoverview",)} **Related Topics**

PDOXWIN.INI and the system registry

Settings that control default properties and application behaviors are now stored in the system registry for Corel Paradox 8.

Previous versions of Corel Paradox allowed users to save these settings in PDOXWIN.INI. Corel Paradox can still use settings from PDOXWIN.INI if Corel Paradox is started with the following command-line setting: **-i PDOXWIN.INI**

When used in this way, the settings in PDOXWIN.INI override those in the system registry.

You can also create a PDOXWIN.INI file using the registry entries as examples. The following table shows corresponding entries in the registry and the .INI file. The registry keys listed below are found at HKEY_CURRENT_USER\SOFTWARE\COREL\PARADOX\8.0.

Key	PFWWIN.INI section	Description
Designer	[Designer]	Specifies designer options
Desktop	[Desktop]	Determines the size and location of the Corel Paradox desktop
Form	[Form]	Specifies ruler and zoom display options in Form windows
History	[History]	Lists the history of directories used in the Project Viewer
ProjectViewer	[Project Viewer]	Determines if the Project Viewer is displayed automatically on startup and whether to display all files, or only specified references (References are specified in the PDOXWORK.INI file. For more information, see PDOXWORK.INI)
Properties	[Properties]	Changes default property settings and application behaviors
Report	[Report]	Specifies the zoom level for reports
Query	[Query]	Specifies options for queries

Note

The following PFWIN.INI settings have changed since version 5.0:

`[Editor Kernel]Tab size` was `[IDE]Tabinc`

`[Editor Window]Save prompt` was `[IDE]Save prompt`

`[Editor Window]Custom size` was `[IDE]UsrDefaultSize`

`[Editor Window]Font name` and `Font size` were `[IDE]Font`, with parameters

{button ,AL(`commandline;iniFiles;systemRegistry;systemRegistryWININI;idh_t_config_pxdliteini;',0, "Defaultoverview",)} Related Topics

Command-line options

Corel Paradox supports the following command-line options:

Option	Behavior
-b	Prevents multiple instances of Corel Paradox from being loaded. If it is already running, the Corel Paradox window is brought to the front. If you try to load Corel Paradox more than once without changing the private directory, an error message tells you that the Borland Database Engine (BDE) could not be initialized.
-c	Starts Corel Paradox with a clear desktop
-d Filename	Specifies an alternate PDOXWORK.INI, which contains folder definitions for use with the Project Viewer. By default, Corel Paradox does not use an .INI file to store settings, but you can create a file named PDOXWORK.INI in your working directory, and Corel Paradox will read settings from it automatically. If you specify a full directory path (one that includes a drive letter and directory) with filename, Corel Paradox uses that file for every working directory you use. However, if you do not include a directory reference and you change your working directory after startup, Corel Paradox then looks for the filename in the new directory. If the file cannot be found, Corel Paradox uses the new directory's PDOXWORK.INI.
-e	Prevents writes to the system registry
-f	Forces writes to the system registry
-i Filename	Specifies an alternate PDOXWIN.INI file, which contains desktop settings. By default, Corel Paradox does not use an .INI file to store settings, but you can create a file named PDOXWIN.INI and Corel Paradox will read settings from it automatically.
-m	Loads Corel Paradox as a minimized application. This is useful if you want to load Corel Paradox but not work with it immediately.
-n	Prevents saving work and private directories on exit
-o Filename	Alternates the BDE configuration file. All BDE-based applications must use the same BDE configuration file when running concurrently.
-p Directory	Starts with a different private directory than the one set in the system registry. Corel Paradox stores its temporary tables in the directory you indicate with this parameter. If you do not indicate a full directory path (one with a drive letter), Corel Paradox looks for the new directory with respect to the Corel Paradox system directory.
-q	Suppresses the Corel Paradox title screen while it is loading
-s	Prevents resizing of the Corel Paradox window
-t	Allows resizing of the Corel Paradox window
-w Directory	Starts Corel Paradox with the specified working directory instead of the one saved in the system registry.
-y	Forces saving work and private directories on exit
StartFile	Opens the specified document and performs its default action. Corel Paradox looks in the current working directory for the specified file unless you include the full directory reference to that file. You can tell Corel Paradox to open a file on startup (for example, a form or report) by typing the name of the file, along with any necessary directory information. This method does not require a special option, but does require the file extension. After loading, Corel Paradox opens the file and performs its default action. For example, forms are displayed in a Form window, scripts are run, and so on.

{button ,AL(`systemRegistry;iniFiles;toUseCommandlineOptions;systemRegistryWININI;idh_t_config_pdoxworkini;idh_t_config_pfwwinini;',0,"Defaultoverview",)} **Related Topics**