How To:

Introduction to scripts and Corel SCRIPT

A script is a computer program that executes a series of instructions with a single command. Generally, scripts are used to automate repetitive tasks or simplify complicated actions, but they can also prompt for user input, display messages, and interact with other applications.

Scripts can significantly increase your productivity with Corel applications by automating repetitive tasks. For example, a script could be used to open a group of files, perform a set of editing actions, or set an application's default properties. In their simplest form, scripts replicate a Corel application's keystrokes, and toolbar, menu, and mouse commands. In a more complex form, scripts can include the commands and constructs of a programming language. For example, you could create a script that only replicates an application's commands once a series of logical requirements have been met.

A Corel script is a Windows text file that lists the Corel SCRIPT commands that will perform a particular task. These instructions are all part of the Corel SCRIPT programming language, which is partially based on Corel application menu commands. For example, the **FileClose** command corresponds to a menu command (click File, Close) on an application's menu system.

The rest of the Corel SCRIPT programming language is based on the BASIC programming language. If you're already familiar with a version of BASIC, you'll find the Corel SCRIPT programming language easy to read and understand.

Computer programming experience isn't a prerequisite for using Corel SCRIPT to create and edit scripts. However, the more knowledge, experience, and desire you have to delve into the mechanics of your Corel application, the more you'll be able to take advantage of the power of Corel SCRIPT.

Your Corel SCRIPT online Help file contains information, from instructions for novice script writers to reference material for experienced script writers and programmers. The following information categories are available in this online Help file:

- Corel SCRIPT introduction and overview
- Programming and running Corel SCRIPT scripts
- Debugging scripts
- Corel SCRIPT programming language syntax reference
- Using the Corel SCRIPT Editor
- Using dialog boxes and the Corel SCRIPT Dialog Editor
- Corel SCRIPT reference information

The amount of information you'll need to know about scripting will depend on the complexity of your scripts.

Note

- Most large computer applications have a built-in programming language of some form but some call their programs macros instead of scripts.
- Not every Corel application supports Corel SCRIPT programming and script files. Click + for a list of Corel applications that support Corel SCRIPT.

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

How To:

What is Corel SCRIPT?

A typical user can take rapid advantage of Corel SCRIPT to automate repetitive tasks with scripts written in the Corel SCRIPT language. However, Corel SCRIPT is more than just a language that you can use to create and run script files; it's also a powerful programming language that can be used as a stand-alone application.

The Corel SCRIPT programming language is based on the BASIC programming language. If you're already familiar with a version of BASIC, you'll find the Corel SCRIPT programming language easy to read and understand.

The Corel SCRIPT application includes:

- the Corel SCRIPT Editor
- the Corel SCRIPT Dialog Editor
- advanced Windows programming features for DLL functions and OLE automation capabilities.

Note

- Not every Corel application supports Corel SCRIPT programming and script files. Click + for a list of Corel applications that support Corel SCRIPT.

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

How To:

Corel SCRIPT programming language

The Corel SCRIPT programming language is made up of two sets of instructions: Corel SCRIPT **application** commands and functions, and Corel SCRIPT **intrinsic** statements, commands, and functions. Partially based on the BASIC programming language, the Corel SCRIPT programming language is easy to read and understand. In the on-line Help, the Corel SCRIPT Reference section provides descriptions of each Corel SCRIPT statement, command, construct, and function, along with other reference information and examples.

Corel SCRIPT application commands

Corel SCRIPT application commands instruct Corel applications that support Corel SCRIPT to perform commands. Each Corel application that supports Corel SCRIPT has a distinct set of commands.

Generally, Corel SCRIPT application command names correspond to the command's menu name preceded by its main menu name. For example, the **EditCut** command is the complement of a menu command (click Edit, Cut) on an application's menu system. Customizing your menu structure doesn't affect Corel SCRIPT command names.

Unlike the **EditCut** command, most application commands require more than just a menu name to have the command carried out. Many menu commands in Corel applications open dialog boxes with dialog controls that require user input values. Since application dialog boxes are not displayed during script execution, the values required for commands that open dialog boxes must be specified with the command in the script file. These specified values are called parameters and usually correspond to dialog box options.

For example, if you wanted to open a file named **myfile.txt** in a Corel application, you might use a command similar to the following:

```
.FileOpen "myfile.txt"
```

where **"myfile.txt"** is a parameter for **FileOpen**.

Corel SCRIPT application commands are often available in more than one application. For example, the **FileNew** command is available in both CorelDRAW and Corel PHOTO-PAINT. Although the **FileNew** command creates a new document in both applications, **FileNew** has different parameter settings in each application. For example, if you used CorelDRAW commands in a script for Corel PHOTO-PAINT, an error would likely occur.

In Corel SCRIPT Help, application statements are in initial caps, such as **FileOpen**, **EditCut**, **FilePrint**.

Corel SCRIPT intrinsic statements

Corel SCRIPT intrinsic statements and functions are based on the BASIC programming language and perform instructions or actions that are not part of a Corel application. For example, Corel SCRIPT intrinsic statements can be used to display a user-defined dialog box, include flow control statements and constructs such as loops, create and manipulate variables, and retrieve information about your computer setup. On their own, Corel SCRIPT intrinsic statements form a powerful programming language. In fact, a script containing only Corel SCRIPT intrinsic statements can be executed even if a supporting Corel application is not running.

In Help, Corel SCRIPT intrinsic statements are in uppercase, such as **LEFT**, **IF**, and **MESSAGEBOX**.

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

How To:

Corel SCRIPT scripts

A Corel SCRIPT script is a Windows text file (**.CSC** extension) that lists commands and instructions to execute.

Application commands

The most basic script file contains only Corel SCRIPT application commands. These application commands are translated by a Corel application that supports scripts and are executed. In other words, the script tells the Corel application what to do.

The following example is a Corel PHOTO-PAINT script that opens a new image, draws a line, and saves a PHOTO-PAINT file with the name **example** (the script lines beginning with an apostrophe are script comments, and aren't executed):

```
WITHOBJECT "CorelPhotoPaint.Automation.6"
    'Creates a new file
    .FileNew 360, 504, 1, 300, 300, 0, 0, 0, -1, -1, -1, -1, 255, 255, 255, 0
    'next 3 lines set default settings
    .SetPaintColor 5, 0, 0, 0, 0
    .PenSettings 17, 10, 0, 0, 0, 0, 0
    .ShapeSettings 0, 2, 10, 0, 0
    'next 3 line draw a line
    .StartDraw 91, 441
    .ContinueDraw 213, 214
    .EndDraw
    'saves the file
    .FileSave "C:\myfolder\example.CPT", 1792, 0
END WITHOBJECT
```

As you can see in the above example, the command names are always placed at the beginning of a line, and are followed by parameter settings. Command lines cannot wrap over a line. You can have more than one command on a line but the commands must be separated by colon (:) as in the following example:

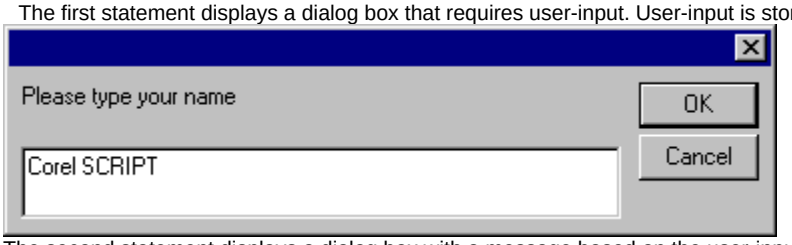
```
.StartDraw 91, 441 : .ContinueDraw 213, 214 : .EndDraw
```

Intrinsic statements

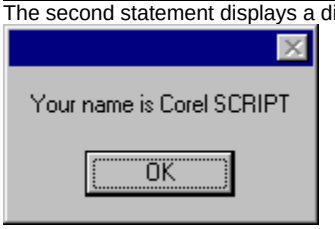
Your script files can also include Corel SCRIPT intrinsic statements, which are based on the BASIC programming language. Intrinsic statements perform instructions or actions that are not part of a Corel application. For example, the following two intrinsic statements each open a dialog box:

```
yourName$ = INPUTBOX("Please type your name")
MESSAGE "Your name is " + yourName$
```

The first statement displays a dialog box that requires user-input. User-input is stored in the variable **yourName$**.



The second statement displays a dialog box with a message based on the user-input to the first dialog box.



Combining application commands and intrinsic statements

The intrinsic statements shown above are a simple example of how to use Corel SCRIPT intrinsic statements on their own. To bring out the real power and flexibility of Corel SCRIPT scripts, you can combine Corel SCRIPT application commands with intrinsic statements. The following example is a script that combines PHOTO-PAINT application commands with Corel SCRIPT intrinsic statements. Intrinsic statements are displayed in boldface, and some application commands and intrinsic statements are preceded by comments:

```
'Create a list of file names called "files"
diskdir$ = INPUTBOX("Please enter the drive letter of your CD-ROM")
DIM files(5) as STRING
```

```
files(1) = "Boxpanel.bmp"
files(2) = "Clamtext.bmp"
files(3) = "Footprnt.bmp"
files(4) = "Fuzzy.bmp"
files(5) = "Gradient.bmp"
'begins the loop
FOR i% = 1 TO 5
WITHOBJECT "CorelPhotoPaint.Automation.6"
    'creates a name for the edited bitmap
    FileName$ = diskdir+":\photopnt\plgbrush\" + files(i)
    'open a bitmap file
    .FileOpen FileName$, -1, -1, -1, -1, 0
    FileName$ = "c:\temp\brushes\brush0" + CSTR(i%) + ".bmp"
    .Imageconvert 1,1,0,0,0,0,0
    .SetPaintColor 5, 0, 0, 0, 0
    .SetPaperColor 5, 255, 0, 0, 0
    'replaces color base on the 3 commands above
    .ColorReplace
    'saves the file with a new name
    .FileSave FileName$, 769, 0
END WITHOBJECT
NEXT i%
```

The above script creates a list of bitmap files, opens each file in a FOR...NEXT loop, converts black & white images to RGB images, replaces white color to red color, and finally saves the edited bitmap file with a new name.

If you're an inexperienced script (or macro) writer, the above script can seem somewhat overwhelming at first, but if you break it down into its basic components (each command and statement), you may find that the script is not as complicated as it looks. With a little practice, and use of the Corel SCRIPT reference online Help, you could be on your way to creating script files in no time.

Corel SCRIPT also comes with sample scripts that you can use to learn more about scripts and how to use commands and statements. The samples can even be used as templates for creating your own scripts.

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

How To:

When should I write a script?

You should consider writing a script any time you are repeating a series of commands in a Corel application that supports Corel SCRIPT. Other times you may be faced with a repetitive task, such as changing a color in a group of Corel documents, and you may say to yourself, "I wish my Corel application could do this for me." Actually, your Corel application can do it, using a script file.

After working with a Corel application for some time, you may begin to see patterns in your work habits. For example, you may repeat actions such as opening the same documents each time you launch your Corel application. In this case, you could create a script to open your files for you. Additionally, if you use a script often enough, you may want to assign a keystroke to it, add it to your menus, or turn it into a button on the toolbar. In other cases, you may just want a script to get you through a tedious task more quickly; once the script is no longer required you can discard it.

As you gain experience with scripts and Corel SCRIPT's powerful programming language, you'll find you can write a script to do almost anything for you in a Corel application that supports scripts. Your only limitation is your imagination.

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

How To:

Executing scripts

Corel SCRIPT scripts can be executed or run, under specific conditions, from any Corel application that supports Corel SCRIPT, or from the Corel SCRIPT Editor. The Corel SCRIPT Editor is a separate application that you use to write, edit, run, and debug script files. Additionally, there are advantages and disadvantages to running a script with your application or with the Corel SCRIPT Editor.

Scripts containing Corel SCRIPT application commands

Any script file that contains application commands for a Corel application must include the **WITHOBJECT** statement. The WITHOBJECT statement directs the executing script to the Corel application to call.

The following example shows how to run a **FileNew** command in Corel PHOTO-PAINT.

```
WITHOBJECT "CorelPhotoPaint.Automation.6"
    .FileNew 360, 504, 1, 72, 72, 0, 0, 1, 0, 0, 0, 19533528, 255, 255, 255, 0
END WITHOBJECT
```

You can have as many commands as you want after the **WITHOBJECT** statement, but the block of commands must end with an **END WITHOBJECT** statement. You can also have Corel SCRIPT intrinsic statements within the **WITHOBJECT** block of commands.

You have the option of running scripts from the Corel SCRIPT Editor or from your Corel application that supports Corel SCRIPT. Not every Corel application supports Corel SCRIPT programming and script files. Click + for a list of Corel applications that support Corel SCRIPT.

When you create or edit a script, you should first run it from the Corel SCRIPT Editor. Executing from the Editor lets you take advantage of the Editor's testing and debugging features that allow you to fine tune script syntax or fix script syntax errors. However, once you are satisfied your script is running properly, you should run your script from the Corel application that uses the application commands. Running from the application can significantly decrease the script's execution time.

From the Editor, you can run a script that contains Corel application commands without having the application open. The script will be executed in the computer memory, in the background. However, in most cases, your script will have to open a file or create a new document before the rest of the application commands can be executed.

You can also nest application commands for different Corel applications within a script. For example, you can create a script that copies an object from PHOTO-PAINT to CorelDRAW:

```
WITHOBJECT "CorelPhotoPaint.Automation.6"
    'Series of PHOTO-PAINT commands to copy an object
    WITHOBJECT "CorelDraw.Automation.6"
        'Series of DRAW commands to paste an object
    END WITHOBJECT
END WITHOBJECT
```

Executing applications commands in the background

If you execute a script from the Corel SCRIPT Editor (or from a Corel application) containing commands for an application that is not already running, Corel SCRIPT attempts to start the application. If the application is started, it is only opened in a portion of the computer memory called the *background*. Applications opened in the background are not visible on the Windows desktop. (Pressing CTRL+ALT+DEL opens the Close Program dialog box indicating active applications, both visible and invisible.) Corel SCRIPT executes commands for applications only open in the background the same way it does for visible applications.

The following should be kept in mind when executing a script for a Corel application that is not already open:

- The first script application command after the **WITHOBJECT** statement must be a .**FileNew** or .**FileOpen** command.

How To: If an error occurs while your script is executing, the hidden application becomes visible.

How To: If the last script application command before the **END WITHOBJECT** statement is not a .**FileClose** (for CorelDRAW 6 or Corel PHOTO-PAINT 6) or .**FileExit** (for CorelFLOW 3 or CorelCAD 1), the hidden application becomes visible after executing the last command.

How To: You can make a hidden application visible by including a .**SetVisible** command in your script.

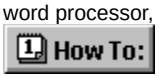
Tip

How To: You can also execute a script by double-clicking on its icon in the Windows Explorer or in a Windows folder.

Note

How To: Scripts containing only intrinsic statements can be run from any Corel application that supports Corel SCRIPT or from the Corel SCRIPT Editor. Generally, a script that contains only intrinsic statements runs faster from the Corel SCRIPT Editor.

Though script files are text files, and can be edited or created with almost any Windows-based text editor or word processor, you must use the Corel SCRIPT Editor if you want to test or debug a script.



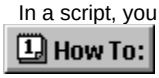
Unlike script files (or macro files) from other companies, Corel SCRIPT files are text only; there is no compiled binary component in the scripts. Before a script is executed, it is compiled internally into a program file.

{button ,AL(`intro_cs;Script_procedures;Script_programming_errors;;;',0,"Defaultoverview",)} Related Topics

Script procedures

A simple Corel SCRIPT script executes in a linear manner; that is, each statement is executed on a line-by-line basis up to the last script statement. In complex scripts, script execution jumps to blocks of statements called procedures.

In a script, you can have three types of procedures:



functions

subroutines

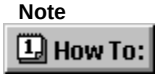
main

Functions and subroutines are groups of Corel SCRIPT statements that are executed when the procedure is called by another Corel SCRIPT statement. Both types of procedures are useful in cases when a group of instructions is to be repeated. The instructions are written once in the script, and can be called from different places within the script or with different parameters. You can have more than one of each type of procedure in a script.

Although, subroutines and functions are both Corel SCRIPT procedures that execute instructions, functions can also be used to return values to a script that can be either assigned to an expression or compared with other expressions.

Any script statement that is not part of a subroutine or a function is considered part of the script's main procedural section. Each script has only one main section.

Note



The maximum number procedures you can have in a script is 125.

{button ,AL(`const;global;call;Variable_availability;Using_functions_subroutines;Executing_script_files;declare;',0,"Defaultoverview",)} Related Topics

How To:

Corel SCRIPT utilities

The Corel SCRIPT application comes with two utility programs to make it easier for anyone from an inexperienced script writer to a professional computer programmer to create, test, and debug scripts.

Corel SCRIPT Editor

The Corel SCRIPT Editor is a separate application that you can use to create, edit, run, test, and debug a Corel SCRIPT script file.

Though script files are text files, and can be edited or created with almost any Windows-based text editor or word processor, you must use the Corel SCRIPT Editor if you want to test or debug a script.

Corel SCRIPT Dialog Editor

In many cases, you'll need to get information from the user before your script performs a desired action. For simple information, you can use the Corel SCRIPT function INPUTBOX to get a string from the user returned to a running script. If you want to provide the user with options and more complex information, such as a list of choices, you can use a dialog box in your script.

Dialog boxes are created using the Corel SCRIPT language. The Corel SCRIPT language features a full set of programming statements to produce dialog boxes which incorporate sophisticated Windows options and features.

The Corel SCRIPT Dialog Editor is a tool to quickly create and edit Corel SCRIPT dialog boxes. Working with the Dialog Editor is similar to using a drawing or painting application: dialog controls are graphic objects which can be inserted, moved, re-sized, and aligned in a dialog box.

Using the Dialog Editor takes the place of creating and editing Corel SCRIPT statements in a script file. It involves a visual approach to creating and editing Corel SCRIPT dialog statements, since a dialog box and the controls within it represent Corel SCRIPT statements. From the Dialog Editor, you can transfer the dialog boxes you've created to the Clipboard, save the dialog boxes as Corel SCRIPT script files, and insert Corel SCRIPT statements into a script.

Note

How To: Unlike script files (or macro files) from other companies, Corel SCRIPT files are text only; there is no compiled binary component in the scripts. Before a script is executed, it is compiled internally into a program file.

{button ,AL(`corel_script_dialog_editor;corel_script_editor;corel_script_editor_basics;corel_script_and_dialog_boxes;;',0,"Defaultoverview",)} Related Topics

Using Dynamic Link Libraries

Corel SCRIPT can call functions and subroutines in Dynamic Link Libraries (DLLs) such as those supplied with Windows, other applications' DLLs, or any custom DLL files. To find out how to use functions in DLLs, you need specific technical reference material. For example, to use the Windows DLLs, you need the *Windows Software Development Kit*.

Before you can use a DLL function or subroutine, you must declare the function using the DECLARE...LIB statement. See the statement's reference for more information and an example.

Warning

You should save or back up essential files and programs before using functions and subroutines in DLL files. Passing an invalid argument to a function can result in a Windows General Protection Fault or unstable system behavior.

Note

The advanced Corel SCRIPT programming feature described above is intended for experienced Windows programmers, and not for beginner or intermediate script writers.

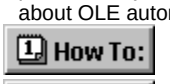
{button ,AL(`declare_lib;getapphandle;getwinhandle;;;',0,"Defaultoverview",)} Related Topics

Corel SCRIPT and OLE automation

Any Corel application that supports Corel SCRIPT provides a programmable OLE automation **object**. The object is used by OLE automation controllers to send Corel SCRIPT application commands to their respective Corel application. For example, Corel SCRIPT DRAW application commands are sent to CorelDRAW.

You can use OLE automation controllers such as Microsoft Visual Basic, Microsoft Excel Visual Basic, and Microsoft Visual C++ (with the Microsoft Foundations Classes) to send commands to applications that support OLE automation objects such as CorelDRAW, and to develop applications using Corel application components.

The *Corel application commands and functions* in this online Help file provide a reference of all available commands and functions in your Corel application. The commands and functions are a part of **automation-enabled objects**. The online Help provides only overview procedural and reference information about programming with OLE automation. For more information about OLE automation, see the following reference sources:



Microsoft Visual Basic Programmer's Guide

Microsoft Excel Visual Basic User's Guide

Microsoft Windows Developer's Kit

Microsoft Office Developer's Kit

Corel SCRIPT Editor

The Corel SCRIPT Editor is an OLE automation controller; that is, you can use it to access objects in other applications that have OLE automation objects. Ordinarily, the Editor is used to access objects in Corel applications that support OLE automation. However, you can also access objects in non-Corel applications. For example, you can call Microsoft Word 6.0 or Microsoft Excel 5.0 using the external names "Word.Basic" or "Excel.sheet.5", respectively.

To access objects in applications that support OLE automation from the Corel SCRIPT Editor, use the WITHOBJECT statement.

Note

For a list of Corel applications that support OLE automation, and their application object names, click

.

The advanced Corel SCRIPT programming feature described above is intended for experienced Windows programmers, and not for beginner script writers.

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

Using Corel applications with OLE automation controllers

To access a Corel application with an OLE automation controller, an object variable must be first defined for the Corel application. Each Corel application that supports OLE automation has one object that can be accessed by a controller. For a list Corel applications that support OLE automation and their respective automation object names, click 

. The following section provides an example of using Microsoft Visual Basic to access Corel PHOTO-PAINT. Other OLE automation controllers may access OLE automation objects with different instructions.

Example

From Visual Basic, the first step is to declare an object variable type. For example,

```
DIM paint AS OBJECT
```

The next step is to assign the application object to the object variable previously declared. In this case, the object variable is **paint,** and the Visual Basic **CreateObject** function with the PHOTO-PAINT object name is used to assign the application object. For example,

```
SET paint = CREATEOBJECT ("CorelPhotoPaint.Automation.6")
```

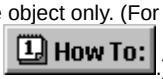
Corel PHOTO-PAINT commands can now be accessed by Visual Basic.

Starting applications

Before an automation controller (for example, Visual Basic) can access a Corel application that supports OLE automation objects, the application must be running. If it is not running, the controller attempts to start it. The application location and path information is stored in the Windows Registry.

You can use the Visual Basic **Set Nothing** command to discontinue an association to a previously declared object.

One object in Corel OLE automation applications

Corel applications that support OLE automation have one object only. (For a list Corel applications that support OLE automation and their respective Corel application object names, click 

.)

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

Using Corel SCRIPT application commands: an example

Once an OLE automation controller has assigned a Corel application object to a variable and made it available, it can use Corel SCRIPT statements, functions, and commands the same way a Corel SCRIPT script uses them.

However, the OLE automation controller can only use Corel application statements. For example, it can use the commands **EditCut** from PHOTO-PAINT and **FileOpen** from DRAW, but it cannot use Corel SCRIPT intrinsic statements and functions such as **FOR...NEXT** or **MESSAGE** or Corel dialog box definition statements. See the Corel SCRIPT programming language for more details about application commands, intrinsic statements, and functions..

The following example, which uses Visual Basic as an automation controller and Corel PHOTO-PAINT as the OLE automation application receiving instructions, creates a Windows bitmap called SQUARE.BMP. The commands in bold are Visual Basic statements; the others are Corel SCRIPT application commands.

```
DIM paint AS OBJECT   'declare on object variable
SET paint = CREATEOBJECT ("CorelPhotoPaint.Automation.6") 'assigning the application object
    .FileNew 216, 288, 4, 72, 72, 0, 0, 1, 12643084, 12, 87753640, 11203980, 255, 255, 255, 0
    .SetPaintColor 5, 51, 255, 0, 0
    .PenSettings 20, 20, 0, 0, 0, 0, 0
    .ShapeSettings 50, 0, 20, -1, 0
    .Rectangle 52, 218, 196, 65
    .FileSave "C:\COREL60\PROGRAMS\square.BMP", 769, 0
SET paint = NOTHING   'discontinues the association to the declared object
```

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

Corel SCRIPT Editor: OLE automation controller

Not only can you use the Corel SCRIPT Editor to edit and debug Corel SCRIPT scripts, but you can use it as an OLE automation controller. Any time you run a script from the Editor that contains Corel SCRIPT application statements, you are using the Editor's OLE automation capabilities to access Corel application commands.

You can also use the editor to create scripts that can call non-Corel applications. For example, you can call Microsoft Word 6.0 or Microsoft Excel 5.0 using the external names "Word.Basic" or "Excel.sheet.5" respectively, with the WITHOBJECT statement.

The following Corel SCRIPT script example creates a new Microsoft Word 6.0 document called NAMEDATE.DOC that contains the user's name and the current date:

```
WITHOBJECT "Word.Basic"
    .FileNew .Template = "Normal", .NewTemplate = 0
    .InsertField .Field = "USERNAME  \* MERGEFORMAT"
    .Insert Chr$(9)
    .InsertDateTime .DateTimePic = "dddd, MMMM dd, yyyy", .InsertAsField = 1
    .FileSaveAs .Name = "NAMEDATE.DOC", .Format = 0, .LockAnnot = 0, .Password = "", .AddToMru
= 1, .WritePassword = "", .RecommendReadOnly = 0, .EmbedFonts = 0, .NativePictureFormat =
0, .FormsData = 0
END WITHOBJECT
```

The Word Basic commands, or any commands from an OLE automation application receiving instructions, must be preceded by a period. In the above example, the Word Basic **FileNew** command became the **.FileNew** command.

Note

How To: You can also execute the script example shown above, or any other script that calls the automation controller from any Corel application that supports Corel SCRIPT. See To run a Corel SCRIPT script from a Corel application for more information.

How To: Before the Corel SCRIPT OLE automation controller can access an OLE automation application (for example, Microsoft Word 6.0), the application that is being accessed must be running. If it is not running, the Corel SCRIPT Editor controller attempts to start it. The application location and path information is stored in the Windows Registry.

How To: After the **END WITHOBJECT** command is executed, the Corel SCRIPT OLE automation controller can no longer access the object declared in the **WITHOBJECT** command until another **WITHOBJECT** command is issued.

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

Documentation syntax conventions

The Corel SCRIPT Language Reference section provides syntax and reference information for each statement, command, and function in Corel SCRIPT.

The following typographic conventions are used in the Corel SCRIPT Language Reference :

Syntax convention example	Description
LBOUND, **IF**, **LCASE**	Corel SCRIPT intrinsic statements appear in boldface, all uppercase.
.FileOpen, .**EditCut**	Corel SCRIPT application commands and functions appear in boldface, with the initial letter of each word in uppercase. The command is preceded by a period. Generally, Corel SCRIPT application command names correspond to the command's menu name preceded by its main menu name. For example, the .**EditCut** command is the complement of a menu command (click Edit, Cut) on an application's menu system.
{%|&|!|#|@|$}	Braces and vertical bars surrounding the variable type suffixes indicate a choice between two or more items. The choice is mandatory if the syntax appears in boldface. If the syntax appears in normal face, use of the suffix is optional.
{ WHILE | UNTIL }	Braces and vertical bars surrounding statement keywords indicate a choice between two or more items. The choice is mandatory if the syntax appears in boldface. If the syntax appears in normal face, syntax use is optional.
`Script code`	Script code appears in this font in examples.
`'Comments`	Script example comments appear with an apostrophe ('). In some cases the **REM** statement is used instead of the apostrophe.
intrinsic parameter	Required parameters for Corel SCRIPT intrinsic statements appear in boldface.
intrinsic parameter	Optional parameters for Corel SCRIPT intrinsic statements appear in normal face.
command parameter	Required parameters for Corel SCRIPT application commands appear in an italic boldface.
command parameter	Optional parameters for Corel SCRIPT application commands appear in italics.

Note

For more details about Corel SCRIPT parameters, see Script parameters.

Corel SCRIPT application command parameters must be separated by commas.

Required parameters are displayed in a boldface while optional parameters are displayed in a normal face. Optional parameters normally appear at the end of a command's syntax.

{button ,AL(`intro_cs;cs_case_sensitive;corel_script_editor_basics;;;',0,"Defaultoverview",)} Related Topics

Script parameters

Parameters are variables, constants, or expressions that provide information to intrinsic statements and application commands. Most Corel application commands use parameters to pass values to commands. The parameters pass values to a Corel application much the same way dialog box controls do. For example, if you wanted your script to open a file named **myfile.txt** in a Corel application, you might use a command similar to the following:

```
.FileOpen "myfile.txt"
```

where **"myfile.txt"** is a parameter for **FileOpen**.

Script parameters for application commands

In the Corel SCRIPT Language Reference section, each application command's syntax is displayed with parameter names, if available. For example, the Corel PHOTO-PAINT command .**SetPaperColor** has the following syntax:

.SetPaperColor .Type = ***long*****, .Color1=*****long*****, .Color2=*****long*****, .Color3=*****long*****, .Color4=*****long***

The strings appearing before the equal signs are the parameter names and the italic strings appearing after the equal signs indicate the variable type accepted by the parameter. (See Corel SCRIPT data type summary for variable details.)

With most Corel SCRIPT application commands, you have options on how to specify parameters. The following examples shows two methods to specify the .**SetPaperColor** command:

```
.SetPaperColor .Type = 5, .Color1 = 255, .Color2 = 0, .Color3 = 0, .Color4 = 0
.SetPaperColor 5, 255, 0, 0, 0
```

The first example uses the parameter names and equal signs while the second simply separates parameters with commas. Using the parameter names makes your script more self-documenting but is not as easy to write. You also have the option to mix the parameter specification for a command. For example:

```
.SetPaperColor 5, 255,.Color2 = 0, .Color3 = 0, .Color4 = 0
```

Once you use a parameter name in an application command, the parameters that follow in that command must also use parameter names.

Optional parameters

In some cases, application parameters are optional; that is, they are not required in the script. For example, in the following CorelFLOW command the last two parameters are optional:

.RotateObject .Angle=***double*****, .Clockwise=*****Boolean*****,** .ObjectName=*string*, .FileNumber=*integer*

In this case, you could specify .**RotateObject** without specifying the third parameter, the fourth parameter, or both, as shown in the examples below (the apostrophe indicates a script comment):

```
.RotateObject 58.3, -1, "poly", 2 'no omitted parameters
.RotateObject 58.3, -1, , 2       'third parameter omitted
.RotateObject 58.3, -1, "poly"    'fourth parameter omitted
.RotateObject 58.3, -1            'third and fourth parameters omitted
```

You can also omit optional parameters when you use the parameter names as shown in the examples below (these examples replicate the previous example set):

```
.RotateObject .Angle=58.3, .Clockwise=-1, .ObjectName="poly", .FileNumber=2   'no omitted
.RotateObject .Angle=58.3, .Clockwise=-1, .FileNumber=2   'third omitted
.RotateObject .Angle=58.3, .Clockwise=-1, .ObjectName="poly"   'fourth omitted
.RotateObject .Angle=58.3, .Clockwise=-1   'third and fourth omitted
```

Using variables with parameters

You can also use variables in place of actual numeric and string values in Corel application commands. A variable is a value place holder whose name points to an address in the computer's memory where that value is stored. For example, the two following commands perform the same way provided that **degree_no** equals 58.3 and **cc** equals -1.

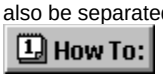
```
.RotateObject 58.3, -1
.RotateObject degree_no, cc
```

Script parameters for intrinsic statements

Intrinsic statements follow the same parameter specification rules as application commands. However, for the most part, intrinsic statements don't use parameter names. Script parameters specify the variable type by using the data type suffix.

Note

Parameters must appear in the order specified in the Corel SCRIPT Language Reference. Parameters must also be separated by commas if specified in the Corel SCRIPT Language Reference.

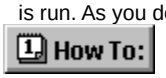


CorelDRAW application commands do not support named parameters.

{button ,AL(`using_variables;Documentation_syntax_conventions;cs_case_sensitive;Corel_SCRIPT_Editor_Basics;Variable_availability;;',0,"Defaultoverview",)} Related Topics

Script programming errors and debugging

Corel SCRIPT scripts are Windows text files; they do not have a compiled binary component. Before a script is executed, it is compiled internally into a program file, and then run. Programming errors can occur both when the script is compiled and when it is run. As you design and run scripts, there are three types of errors that can occur:

 Compilation errors

 Run-time errors

 Logic errors

Compilation errors

Compilation errors prevent Corel SCRIPT from compiling a script into machine instructions. Compilation errors are easy to find since Corel SCRIPT detects them, and notes them in the Compiler Output Window in the Corel SCRIPT Editor.

The most common compilation errors include:

 typographic errors such as misspelling variable names

 missing opening or closing brackets

 missing a required parameter for a statement or a function

 missing a corresponding closing statement, for example, omitting the FOR statement when using the FOR...NEXT construct

 incorrect usage of a statement or function

The compiler reports a maximum of ten errors. Once the compiler finds eleven errors, it stops. You must correct the errors, and re-compile.

Run-time errors

Run-time errors occur when a script is run. These errors are generated when scripts produce bad or illegal values, or try to run an impossible operation. You can design your scripts with run-time errors in mind. For example, an UNABLE TO DELETE FILE run-time error might occur when you try to remove a file. Since you may not know the restrictions on the file you're trying to remove, you should write an error handling routine to trap for this possibility and have your script act accordingly. See Trappable Error Codes or the ON ERROR command for more information.

The most common run-time errors include:

 division by 0

 variable type mismatch

 file access errors

 variable overflow

Logic errors

Logic errors are the hardest to find; the only indication of a logic error may be a bad value or an unexpected result. The Corel SCRIPT Editor cannot tell you when a logic error is present, so it is up to you to test for and find these problems. To help you, the Corel SCRIPT Editor provides debugging tools to trace through scripts more carefully, tracking the values of variables and function parameters, and following the flow of execution. For more information about these tools, see Corel SCRIPT Editor debugging features .

Tips and troubleshooting

 From the Corel SCRIPT Editor you can open Corel SCRIPT online Help to a command's or statement's syntax reference by placing the insertion point in the keyword you want help for and pressing F1.

 You should run your scripts from the Corel SCRIPT Editor until they run error free. Executing scripts from the Editor lets you take advantage of its testing and debugging features. However, once you are satisfied a script is running properly, you should run the script from a Corel application for significant time savings.

If Corel SCRIPT attempts to execute an application's commands and it is not already running, Corel SCRIPT attempts to start the application in a portion of the computer memory called the *background*. The best way to ensure that all applications running in the background are closed after script execution is to have the script close all its documents. If your script is not closing the open documents and leaving the application to run in the background, press CTRL+ALT+DEL to open the Close Program dialog box to close the invisible applications.

In some cases, you can have both a visible and an invisible instance of an application running in the background. In this case, Corel SCRIPT executes the application commands on the application instance which was first started. This can result in confusion if the invisible application was started first and you expect the Corel SCRIPT application commands to be executed in the visible instance of the application. Press CTRL+ALT+DEL to open the Close Program dialog box to close the instance of the application you want to close.

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

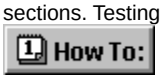
Script planning and designing tips

It is a good idea to plan the script before you begin writing. Keep the following criteria in the mind:

How much of any given process do you want to automate? Will the script be useful for one document and of not much use in another? Which input boxes and dialog boxes are needed to obtain information while the script is running?

Building error checking in your scripts prevents unexpected results. For example, you can tell a script to terminate if the intermediate script results are not suitable. You can also keep the user from entering inappropriate information, such as an inappropriate value in a dialog box.

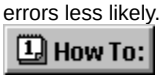
Build and test your scripts in parts. You're apt to make fewer mistakes when you build your scripts in smaller sections. Testing and debugging smaller script sections is easier than trying to trace errors through a large complicated script.



Take some time to lay out the steps a script will perform. Use plain language to describe the problem to be solved and the script solution in remark statements (REM) at the beginning of the script. Use remark statements at the beginning of each section to describe what that section will do.

Tips

You can use the PHOTO-PAINT Command Recorder to record your PHOTO-PAINT actions. They can then be saved as a script, which you can edit and customize. For example, you could add looping constructs or conditional statements to the recorded script. Using the Command Recorder saves you the time it takes to input the PHOTO-PAINT commands and makes syntax errors less likely.

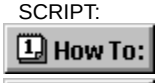


You should run your scripts from the Corel SCRIPT Editor until they run error free. Executing scripts from the Editor lets you take advantage of its testing and debugging features. However, once you are satisfied a script is running properly, you should run it from a Corel application to save time.

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

Corel SCRIPT is not case sensitive

When a Corel SCRIPT script is compiled, the names of all constants, variables, functions, and subroutines are converted to uppercase. Therefore, you cannot have a variable called **Left**, for example, because when converted to uppercase, it would conflict with the Corel SCRIPT function called **LEFT**. The following commands are all interpreted the same way by Corel SCRIPT:



LOOP

loop

lOOp

Strings surrounded by double quotes are not converted when a script is compiled and executed.

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

Script files

A Corel SCRIPT script is a text file (**.CSC** extension) that can be edited or created with almost any Windows-based text editor or word processor. However, to test or debug a script, you should use the Corel SCRIPT Editor.

Since script files are text files, you can easily share scripts with other Corel application users by copying script files to floppy disks or shared networks.

Sample script files

Your Corel application comes with sample scripts. You can use these scripts as they are, or you can modify them to work better for you. You can also use them as a foundation for creating your own scripts.

The sample scripts are available in the following folders (based on a typical Corel installation):

C:\corel60\scripts	The scripts in this folder are not application specific; that is, they do not send instructions to Corel applications.
C:\corel60*application*	The scripts in this folder are application specific; that is, they send instructions to Corel applications to perform actions. ***application*** refers to the application's folder. For example, CorelDRAW scripts are saved in the **C:\corel60\draw** folder.

You can also save your own scripts in the folders noted above.

Notes


Remark statements for each sample script are available at the top of each script

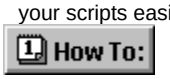

Unlike script files (or macro files) from other companies, Corel SCRIPT files are text only; they contain no compiled binary component. Before a script is executed, it is compiled internally into a program file.


If a script's first line, second line, or both are REM statements, the comments are displayed in a Corel application's Run Script dialog box if the script is specified. The same two REM statements are also displayed in the status bar if the script is assigned to a tool bar button.

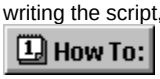
{button ,AL(`cs_custom;intro_cs;Variable_availability;;;',0,"Defaultoverview",)} **Related Topics**

Formatting a script

Scripts that are formatted are easier to follow. Formatted scripts also help you to find and fix errors. Here are some tips to make your scripts easier to read:


Use remarks statements to document your scripts. At the beginning of the script, describe what the script does, how to use it, and give examples of its usage. See REM for more information about using remarks.


Give your scripts meaningful names. You are not limited to the DOS file standard of 8 characters and 3 characters. For example, instead of using **sum.csc,** use **National Output Total.csc**. It is more descriptive and, six months after writing the script, you will still remember what it does.


Give your variables meaningful names. You should also capitalize each word after the first or use the underscore character to separate words. While **x**, **y**, and **z** may sound like good variable names, they don't reveal anything about what is supposed to be stored in the variable. An example of better names include **userMaleName**, **character_Total**, **numberOfPicas** and **typeArray**.

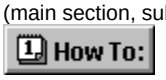

Group commands related to a certain function in the same area, and separate the groups using in blank lines. For example:

```
'Creates a new file
.FileNew 360, 504, 1, 300, 300, 0, 0, 0, -1, -1, -1, -1, 255, 255, 255, 0

'next 3 lines set default settings
.SetPaintColor 5, 0, 0, 0, 0
.PenSettings 17, 10, 0, 0, 0, 0, 0
.ShapeSettings 0, 2, 10, 0, 0

'next 3 lines draw a line
.StartDraw 91, 441
.ContinueDraw 213, 214
.EndDraw

'saves the file
.FileSave "C:\myfolder\example.CPT", 1792, 0
```

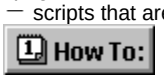

When a set of statements lies within a functional area, indent the set to show that they form a subset. For example, with a **FOR...NEXT** structure, indent the lines after the **FOR** statement.

```
FOR a% = 1 To 4
    [statements]
    FOR b% = 1 To 4
        [statements]
        FOR c% = 1 To 4
            [statements]
        NEXT c%
    NEXT b%
NEXT a%
```


It's a generally accepted programming convention to put declaration statements at the beginning of a procedure (main section, subroutine , or functions).


Define any frequently used numbers or strings as constants. This way, if the value of the constant changes, you need only change it at the top of the script, rather than throughout the script, everywhere it is used. Constants and variables should be defined at the beginning of the procedure in which they are used.


Limit the use of the GOTO statement. When you use the GOTO function, make sure that you don't create spaghetti code
— scripts that are impossible to trace through, because they repeatedly jump from one area of the script to another.


Your function and subroutines procedures should be self-contained. A variable required only within a procedure should be a local or static variable. Following this advice can make your procedures more modular, enabling you to copy them to other scripts with limited customization.

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

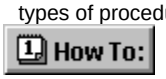
Using constants

In Corel SCRIPT, an expression is a combination of numbers, strings, variables, constants, functions, and operators that return a result. Results can be a number, string, Boolean (TRUE or FALSE), or one of Corel SCRIPTs other data types which can be assigned to a variable.

A constant is an expression that does not change for the duration of a script run. Using a constant instead of a variable ensures you don't accidentally change a value. For example, for some mathematical equations you may want to create the base of the natural logarithm (*e*) or *pi*. To create these two constants, the following lines would be inserted into your script file:

```
CONST NATURAL_LOG# = 2.71828182845 'creates a constant for the base of the natural logarithm
GLOBAL CONST PI# = 3.14 ' creates a global constant for pi
```

The availability of a constant is dependent on the procedure the script is executing. Corel SCRIPT scripts are comprised of three types of procedures:



main instruction or procedural section

functions (more than one can exist)

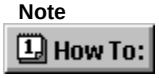
subroutines (more than one can exist)

The following explains the levels of constant availability in Corel SCRIPT:

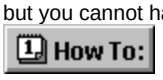
Global constants are available anywhere in a running script but they and their values cease to exist when the script stops running. Global constants are created in the main section of a script and cannot be created within a subroutine or a function. However, they can be used in the execution of any subroutine or function.

Local constants are available in the procedure in which they are declared. If declared in a subroutine or function, a local constant ceases to exist after the procedure finishes execution and is re-created the next time the subroutine or function is called.

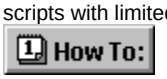
Note



You can have constants with the same name in a script but they cannot exist in the same script procedure (main section, functions, subroutines). For example, you can have a constant called ABC in a function and in the main section of a script but you cannot have two ABC constants in the main section of a script.



Your function and subroutines procedures should be self-contained. A constant required only within a procedure should be a local constant. Following this advice can make your procedures more modular, enabling you to copy them to other scripts with limited customization.



It's a generally accepted programming convention to put constant declaration statements at the beginning of a script's main section, subroutines, or functions.

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

Using arrays

An array is a variable type containing a group of values of the same data type in an ordered list format. For example, the following Corel SCRIPT statements declare and assign values to the string array color:

```
DIM color$(5)
color$(1) = "black"
color$(2) = "red"
color$(3) = "white"
color$(4) = "blue"
color$(5) = "green"
```

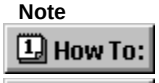
To use an array, you must address a specific element of the array. For example, to use the fourth string in the array mentioned above, you would use the variable **color$(4**).

Arrays are most useful in defining control values in Corel SCRIPT dialog list boxes and with the FOR...NEXT statements.

In the following example, the FOR...NEXT loop is used to assign the numbers 1 through 50 to the corresponding elements of a integer array:

```
DIM numberArray%(50)
FOR i% = 1 TO 50
    numberArray%(i%) = i%
NEXT i%
```

Note



Arrays are created with the DIM (for "dimension") statement.

Arrays can only hold one data type.

You can't change the number of elements in an array once it has been declared.

See Multi-dimensional arrays to create arrays of more than one dimension.

{button ,AL(`listbox_example;FOR_NEXT;DIM;LBOUND;UBOUND;multi_dimensional_arrays;',0,"Defaultoverview",)}
Related Topics

Example

Multi-dimensional arrays

Corel SCRIPT also features multi-dimensional arrays to create tables with more than one column.

Syntax for multi-dimensional arrays

DIM array_name{%|&|!|#|@|$} (l_bound TO u_bound, l_bound TO u_bound, ...)
DIM array_name(l_bound TO u_bound, l_bound TO u_bound, ...) AS type

Syntax	Definition
array_name{%|&|!|#|@|$}	Specifies the name of the array and follows the Corel SCRIPT naming convention. A type-declaration suffix must follow the name in the case of an array.
array_name	Specifies the name of the array and follows the Corel SCRIPT naming convention.
u_bound	The upper bound of the array expressed as an integer. If you do not use a TO clause to specify the number of array elements, the default (1 TO **u_bound**) is used.
l_bound	The lower bound of the array expressed as an integer. If you do not use a TO clause to specify the number of array elements, the default (1 TO **u_bound**) is used.
type	Declares the variable's or array's type with a type declaration name.

Note

How To:

Multi-dimensioned arrays have a limit of five dimensions.

{button ,AL(`using_arrays;dim;lbound;ubound;;',0,"Defaultoverview",)} Related Topics

Example for multi-dimensional arrays

The three following examples create 6-by-6 arrays containing 36 elements:

```
DIM array1$(6, 6)
DIM array2!(-2 TO 3, 2 TO 7)
DIM arr%(0 TO 5, 0 TO 5)
```

We'll use the last example to show how you can use multi-dimensional arrays. If you were to display **arr%** as table it would look like this:

1, 1	1, 2	1, 3	1, 4	1, 5	1, 6
2, 1	2, 2	2, 3	2, 4	2, 5	2, 6
3, 1	3, 2	3, 3	3, 4	3, 5	3, 6
4, 1	4, 2	4, 3	4, 4	4, 5	4, 6
5, 1	5, 2	5, 3	5, 4	5, 5	5, 6
6, 1	6, 2	6, 3	6, 4	6, 5	6, 6

The cells represent array elements and the numbers represent the array index numbers or subscripts. For example, the cell (5, 3) represents **arr%(5, 3)** which represents two integer variables.

The following example creates Pascal's Triangle in a message dialog box using a multi-dimensional array and the FOR...NEXT loop. (Pascal's Triangle is a triangle array of integers in which each number is the sum of the numbers above it in the preceding row. The apex of the triangle is 1.)

```
DECLARE FUNCTION factorial%(a%) 'create an integer formula
DIM arr%(0 TO 5, 0 TO 5)   'this is a 6 by 6 array

FOR i% = 0 TO 5
    FOR j% = 0 TO 5
        arr(i, j) = factorial(i) / (factorial(j)*factorial(i-j))
    NEXT j
NEXT i
'
FOR i% = 0 TO 5
    FOR j% = 0 TO 5
        IF j <= i THEN mess$ = mess$ + CHR(9) + CSTR(arr(i, j))
    NEXT j
    mess$ = mess$ + CHR(13)
NEXT i
MESSAGE mess$
'
FUNCTION factorial%(a%)
    temp% = 1
    IF a > 0 THEN
        FOR lop% = 1 TO a
            temp% = temp% * lop
        NEXT lop
    END IF
    factorial = temp
END FUNCTION
```

{button ,AL(`example_dim;dim;ubound;lbound;;using_variables;using_arrays;',0,"Defaultoverview",)} Related Topics

Using user-defined functions and subroutines

If you have a group of instructions that will be repeated in a script, create a user-defined function or subroutine for those instructions. The instructions are written once in the script, and can be called from different places within the script. If the instructions you want to repeat are changed, the changes take effect everywhere.

Although, user-defined subroutines and functions are both Corel SCRIPT procedures that execute instructions, functions can also be used to return values to a script that can be either assigned to an expression or compared with other expressions.

Before you can create a user-defined function or subroutine, you must declare it. To declare a function, use the following syntax at the beginning of your script:

```
DECLARE FUNCTION CustomFunction%(param1$, param2%)
```

In the example, the first parameter is a string and the second parameter is an integer. The function returns an integer.

To declare a subroutine, use the DECLARE SUB statement, as shown in the example below.

```
DECLARE SUB CustomSubroutine(param1$, param2%)
```

Although you do not use parentheses when you call a subroutine in the script itself, you must use them when you declare the subroutine.

After declaring the functions and subroutines, write the main section of the script. Use the standard syntax for functions and subroutines when you call user-defined functions and subroutines in the script. The code for the user-defined functions and subroutines should be placed after the last line of the main section of the script.

The following example shows a simple script that uses both a user-defined function and a subroutine.

```
REM Description: A script that demonstrates user-defined functions

' First, the function and subroutine must be declared.
DECLARE FUNCTION CustomFunction% (Mystring$, int%)
DECLARE SUB CustomSubroutine (Mystring$, int%)

' This is the main body of the script.  First, call
' CustomSubroutine to assign values to the two variables.
CustomSubroutine Mystr$, num%
' Now call CustomFunction to perform the operation on the
' two variables and return the value.
ret% = CustomFunction%(Mystr$, num%)
' This is the end of the main body of the script.  Now we
' can add the code for the custom routines.

' CustomFunction - takes a string parameter and a number
' The function multiplies the ANSI value of the first character
' of the string times the number and returns the result.
FUNCTION CustomFunction% (Mystring$, int%)
    ' To specify the return value, use the name of the
    ' function as if it were a variable.  By assigning
    ' the result of the calculation to the function
    ' name, the script system knows what the return value
    ' should be.
    CustomFunction% = ASC(Mystring$) * int%

END FUNCTION
' The END FUNCTION statement tells the script system that this
' is the end of the function.  The script then continues to
' run at the next statement after the function call.

' CustomSubroutine - takes a string parameter and a number.
' The subroutine assigns a literal string to the string parameter
' and a numeric value to the number variable.
SUB CustomSubroutine (Mystring$, int%)
    ' Assigning the string to the parameter actually
    ' modifies the variable that was passed to the subroutine.
    Mystring$ = "Test one"
    ' The same is true for the numeric variable.
    int% = 10
END SUB
' The END SUB statement, like END FUNCTION, tells the script
' system to return to the calling routine.
```

Note

Your function and subroutines procedures should be self-contained; that is, a variable only required within a procedure should be a local or static variable. Following this advice can make your procedures more modular, enabling you to copy them to other scripts with limited customization.

{button ,AL(`declare;function_end_function;sub_end_sub;call;;;;;',0,"Defaultoverview",)} **Related Topics**

Example

BEGIN DIALOG...END DIALOG

A user-defined dialog box must begin with the BEGIN DIALOG statement and close with the END DIALOG statement.

The BEGIN DIALOG statement is followed by a series of statements that define a dialog. The series of statements insert dialog controls into the dialog box. Except for remarks statements, the only statements that can appear between BEGIN DIALOG and END DIALOG are the dialog control statements. The END DIALOG statement closes the definition of the user-defined dialog.

A user-defined dialog can be changed by editing the statements between the BEGIN DIALOG and END DIALOG statements. An alternative to editing the statements is to use the Corel SCRIPT Dialog Editor. The Corel SCRIPT Dialog Editor is a tool to quickly create and edit Corel SCRIPT dialog boxes. Working with the Dialog Editor is similar to using a drawing or painting application: dialog controls are graphic objects which can be inserted, moved, re-sized, and aligned in a dialog box.

The BEGIN DIALOG and END DIALOG statements on their own cannot display a dialog box and hold return values during a Corel SCRIPT script run. Use the DIALOG statement to display the dialog box.

Syntax

BEGIN DIALOG Identifier Left%, Top%, **Width**%**, Height**%**,** Text$
[dialog control statements]
END DIALOG

Syntax	Definition
Identifier	Name assigned to the dialog box sequence.
Left%	Distance in dialog units from the dialog box's left border to the left side of the monitor's display area. If both Left and Top are omitted, the dialog box is be centered on the screen.
Top%	Distance in dialog units from the dialog box's top border to the top side of the monitor's display area. If both Left and Top are omitted, the dialog box is be centered on the screen.
Width%	Dialog box width in dialog units.
Height%	Dialog box height in dialog units.
Text$	Label displayed in the dialog box's title bar.
[dialog control statements]	The combination of statements that define a dialog box. Can include any Corel SCRIPT statement that inserts a dialog control into a dialog box.

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

Example

CANCELBUTTON

Adds a Cancel button to a dialog box. Dialog boxes only close when a push button (including the OK button and or Cancel button) is pressed or when the Close Dialog button () is pressed. You should try to include at least an OK button and a Cancel button in each dialog box to make the dialog easier to use.

Syntax
CANCELBUTTON Left%**, Top**%**, Width**%**, Height**%

Syntax	Definition
Left%	Distance in dialog units from the inside of the dialog box's left border to the left side of the check box.
Top%	Distance in dialog units from the bottom of the dialog box's title bar to the top of the check box.
Width%	Cancel button width in dialog units.
Height%	Cancel button height in dialog units.

Returns to dialog box	Condition
2	Pressing the CANCEL button to close the dialog box.

Note

Pressing the Close Dialog button () is the same as pressing the Cancel button; both return 2 to the dialog box.

Detail

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

Example

CHECKBOX

Adds a check box to a dialog box.

Syntax
CHECKBOX Left%, Top%, Width%, Height%, Text$, Value%

Syntax	Definition
Left%	Distance in dialog units from the inside of the dialog box's left border to the left side of the check box.
Top%	Distance in dialog units from the bottom of the dialog box's title bar to the top of the check box.
Width%	Check box width in dialog units.
Height%	Check box height in dialog units.
Text$	Label displayed to the right of the check box. Placing an ampersand (&) before a character provides a keyboard shortcut to selecting the check box.
Value%	**Value** is the variable that holds the return value that corresponds to the state of the check box. Optionally, you can use **Value** to set the default state of the check box.

Returns and Defaults	Condition
0	Check box is disabled, and empty.
1	Check box is enabled, and displays a check mark.
2	Grayed check box. Gray filling a checkbox indicates that a multiple selection contains a mix of property values. For example, selecting text that uses different fonts.

Detail

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

COMBOBOX

Adds a combo box to a dialog box.

Syntax
COMBOBOX Left%**, Top**%**, Width**%**, Height**%**, Array**$**, Value**$

Syntax	Definition
Left%	Distance in dialog units from the inside of the dialog box's left border to the left side of the combo box.
Top%	Distance in dialog units from the bottom of the dialog box's title bar to the top of the combo box.
Width%	Combo box width in dialog units.
Height%	Combo box height in dialog units.
Array$	A one-dimension array containing a string list. The array must be dimensioned before the dialog box in the script.
Value$	**Value**$ is a string variable that holds the return value that corresponds to the selected combo box entry or the user-entered text. Optionally, you can use **Value**$ to set the default selection in the combo box.

Returns and Defaults	Condition
a string	Returns a string that corresponds to the array element selected or the user-entered text.

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

DIALOG

Displays a user-defined dialog box, using a dialog box definition established earlier in the script.

Syntax
DIALOG(Identifier)

Syntax	Definition
Identifier	Name assigned to the dialog box sequence.

Returns	Condition
1	Pressing the OK button to close the dialog box.
2	Pressing the CANCEL button to close the dialog box. Pressing the Close Dialog button (") is the same as pressing the Cancel button; both return 2 to the dialog box.
an integer from 3 to n	The integer corresponds to the push button selected with the first push button being equal to 3. The second push button is equal to 4, and so on. The last pushbutton is equal to (2 + **n**). The order of the push buttons is determined, not by their placement within the dialog box, but by the order in which they are listed in the script.

Note

- Corel SCRIPT dialog boxes are modal which means that the running script cannot continue until the dialog box is closed.

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

DDCOMBOBOX

Adds a drop-down combo box to a dialog box.

Syntax
DDCOMBOBOX Left%**, Top**%**, Width**%**, Height**%**, Array**$**, Value**$

Syntax	Definition
Left%	Distance in dialog units from the inside of the dialog box's left border to the left side of the drop-down combo box.
Top%	Distance in dialog units from the bottom of the dialog box's title bar to the top of the drop-down combo box.
Width%	Drop-down combo box width in dialog units.
Height%	Drop-down combo box height in dialog units when opened.
Array$	A one-dimension array containing a string list. The array must be dimensioned before the dialog box in the script.
Value$	**Value**$ is a string variable that holds the return value that corresponds to the selected drop-down combo box entry or the user-entered text. Optionally, you can use **Value**$ to set the default selection in the drop-down combo box.

Returns and Defaults	Condition
a string	Returns a string that corresponds to the array element selected or the user-entered text.

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

DDLISTBOX

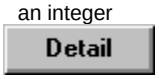
Adds a drop-down list box to a dialog box.

Syntax

DDLISTBOX Left%**, Top**%**, Width**%**, Height**%**, Array**$**, Value**%

Syntax	Definition
Left%	Distance in dialog units from the inside of the dialog box's left border to the left side of the drop-down list box.
Top%	Distance in dialog units from the bottom of the dialog box's title bar to the top of the drop-down list box.
Width%	Drop-down list box width in dialog units.
Height%	Drop-down list box height in dialog units when opened.
Array$	A one-dimension array containing a string list. The array must be dimensioned before the dialog box in the script.
Value%	**Value** is a variable that holds the return value that corresponds to the selected drop-down list box entry. Optionally, you can use **Value** to set the default selection in the drop-down list box.

Returns and Defaults	Condition
an integer	Returns an integer that corresponds to the array element selected.



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

GROUPBOX

Adds a group box to a dialog box.

Syntax
GROUPBOX Left%**, Top**%**, Width**%**, Height**%**, Text**$

Syntax	Definition
Left%	Distance in dialog units from the inside of the dialog box's left border to the left side of the group box.
Top%	Distance in dialog units from the bottom of the dialog box's title bar to the top of the group box.
Width%	Group box width in dialog units.
Height%	Group box height in dialog units.
Text$	Label displayed at the top of the group box.

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

LISTBOX

Adds a list box to a dialog box.

Syntax
LISTBOX Left%**, Top**%**, Width**%**, Height**%**, Array, Value**%

Syntax	Definition
Left%	Distance in dialog units from the inside of the dialog box's left border to the left side of the list box.
Top%	Distance in dialog units from the bottom of the dialog box's title bar to the top of the list box.
Width%	List box width in dialog units.
Height%	List box height in dialog units.
Array	A one-dimension array containing a string list. The array must be dimensioned before the dialog box in the script.
Value%	**Value** is a variable that holds the return value that corresponds to the selected list box entry. Optionally, you can use **Value** to set the default selection in the list box.

Returns and Defaults	Condition
an integer	Returns an integer that corresponds to the array element selected.

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

OKBUTTON

Adds an OK button to a dialog box.

Dialog boxes only close when a push button (including the OK button and cancel button) is pressed, or when the Close Dialog button (▪) is pressed. You should try to include at least an OK button and a Cancel button in each dialog box to make the dialog easier to use.

Syntax

OKBUTTON Left%**, Top**%**, Width**%**, Height**%

Syntax	Definition
Left%	Distance in dialog units from the inside of the dialog box's left border to the left side of the OK button.
Top%	Distance in dialog units from the bottom of the dialog box's title bar to the top of the OK button.
Width%	OK button width in dialog units.
Height%	OK button height in dialog units.

Returns to dialog box	Condition
1	Pressing the OK button to close the dialog box.

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

OPTIONGROUP

Marks the beginning of a series of OPTIONBUTTONs in a script file. The option button statements are positioned directly below the OPTIONGROUP statement and must be together without any intervening statements (remarks excluded).

Syntax
OPTIONGROUP Value%

Syntax	Definition
Value%	**Value** is the name of the option group. It is also a variable that holds the return value that corresponds to the selected option button within the group. Can also be used to set the default enabled button.

Returns	Condition
an integer from 0 to n	The integer corresponds to the option button selected with the first option button in the dialog box definition being equal to 0. The second option button is equal to 1, and so on. The last option button is equal to **n**. The order of the option buttons is determined, not by their placement within the dialog box, but by the order in which they are listed in the script.

Detail

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

OPTIONBUTTON

Adds an option button to a dialog box.

Only one option button in a group can be selected. Its value (0 through n) is returned in the option group variable. The option button statements are positioned directly below the OPTIONGROUP statement in a script. Additionally, the option button statements for each option group must be together without any intervening statements (remarks excluded).

Syntax

OPTIONBUTTON Left%, Top%, Width%, Height%, Text$

Syntax	Definition
Left%	Distance in dialog units from the inside of the dialog box's left border to the left side of the option button.
Top%	Distance in dialog units from the bottom of the dialog box's title bar to the top of the option button.
Width%	Option button width in dialog units.
Height%	Option button height in dialog units.
Text$	Label displayed to the right of the option button. Placing an ampersand (&) before a character provides a keyboard shortcut to selecting the option button.

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

PUSHBUTTON

Adds a push button to a dialog box.
Dialog boxes only close when a push button (including the OK button and Cancel button) is pressed, or when the Close Dialog button (▪) is pressed. You should try to include at least an OK button and a Cancel button in each dialog box to make the dialog easier to use.

Syntax
PUSHBUTTON Left%**, Top**%**, Width**%**, Height**%**, Text**$

Syntax	Definition
Left%	Distance in dialog units from the inside of the dialog box's left border to the left side of the push button.
Top%	Distance in dialog units from the bottom of the dialog box's title bar to the top of the push button.
Width%	Push button width in dialog units.
Height%	Push button height in dialog units.
Text$	Label displayed on the push button. Placing an ampersand (&) before a character provides a keyboard shortcut to selecting a push button.

Returns to dialog box	Condition
an integer from 3 to n	The integer corresponds to the push button selected with the first push button in the dialog box definition being equal to 3. The second push button is equal to 4, and so on. The last pushbutton is equal to (2 + **n**). The order of the push buttons is determined, not by their placement within the dialog box, but by the order in which they are listed in the script.
	The value 1 is reserved for the OK button and the value 2 is reserved for the Cancel button.

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

TEXT

Adds a text to a dialog box.

Syntax
TEXT Left%**, Top**%**, Width**%**, Height**%**, Text**$

Syntax	Definition
Left%	Distance in dialog units from the inside of the dialog box's left border to the left side of the text.
Top%	Distance in dialog units from the bottom of the dialog box's title bar to the top of the text.
Width%	Text width in dialog units.
Height%	Text height in dialog units.
Text$	Static text to display in dialog box.

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

TEXTBOX

Adds text box to a dialog box.

Syntax
TEXTBOX Left%**, Top**%**, Width**%**, Height**%**, Text**$

Syntax	Definition
Left%	Distance in dialog units from the inside of the dialog box's left border to the left side of the text box.
Top%	Distance in dialog units from the bottom of the dialog box's title bar to the top of the text box.
Width%	Text box width in dialog units.
Height%	Text box height in dialog units.
Text$	A string variable that is assigned the text the user enters into the text box. Can also be used to set the default text in the text box.

Returns and Defaults	Condition
a string	Corresponds to the text the user enters into the text box or the default text.

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

SPINCONTROL

Adds a spin control to a dialog box.

Syntax
SPINCONTROL Left%**, Top**%**, Width**%**, Height**%**, Value**%

Syntax	Definition
Left%	Distance in dialog units from the inside of the dialog box's left border to the left side of the spin control.
Top%	Distance in dialog units from the bottom of the dialog box's title bar to the top of the spin control.
Width%	Spin control width in dialog units.
Height%	Spin control height in dialog units.
Value%	**Value** is the a variable that holds the return value that corresponds to the number in the spin control. Optionally, you can use **Value** to set the default value of the spin control. **Value** returns an integer that ranges from -32,767 to 32,767.

Returns and Defaults	Condition
a numeric variable	Returns a number that corresponds to the value selected or entered into the spin control.

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

IMAGELISTBOX

Adds an image list box to a dialog box. The image list box can display Windows bitmaps (.BMP and .RLE files).

Syntax

IMAGELISTBOX Left%**, Top**%**, Width**%**, Height**%**, Array**$**, Value**%

Syntax	Definition
Left%	Distance in dialog units from the inside of the dialog box's left border to the left side of the image list box.
Top%	Distance in dialog units from the bottom of the dialog box's title bar to the top of the image list box.
Width%	Image list box width in dialog units.
Height%	Image list box height in dialog units.
Array$	A one-dimension array containing a string list of graphic files (and their paths). The array must be dimensioned before the dialog box in the script.
Value%	**Value** is a variable that holds the return value that corresponds to the selected image list box entry. Optionally, you can use **Value** to set the default selection in the image list box.

Returns and Defaults	Condition
an integer	Returns an integer that corresponds to the array element selected.

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

IMAGE

Adds a static image box to a dialog box. The image control can display Windows bitmaps (.BMP and .RLE files).

Syntax

IMAGE Left%**, Top**%**, Width**%**, Height**%**, Value**$

Syntax	Definition
Left%	Distance in dialog units from the inside of the dialog box's left border to the left side of the image box.
Top%	Distance in dialog units from the bottom of the dialog box's title bar to the top of the image box.
Width%	Image box width in dialog units. Your selected image is scaled to fit in the image box.
Height%	Image box height in dialog units. Your selected image is scaled to fit in the image box.
Value$	A string or string variable that specifies the graphic (and it's full path) to display in the image box.

Note

- You can insert Windows bitmaps (.BMP and .RLE files) into an image control.
- While editing a dialog box in the Corel SCRIPT Dialog Editor, a placeholder image is displayed in the dialog box.

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

HELPBUTTON

Adds a help button to a dialog box.

Syntax

HELPBUTTON Left%**, Top**%**, Width**%**, Height**%**, Text**$**, Value**%

Syntax	Definition
Left%	Distance in dialog units from the inside of the dialog box's left border to the left side of the help button.
Top%	Distance in dialog units from the bottom of the dialog box's title bar to the top of the help button.
Width%	Help button width in dialog units.
Height%	Help button height in dialog units.
Text$	A string or string variable that specifies the help file (and it's path) that you want to open.
Value%	Specifies the default topic to display when the help file opens. You should use the context number ID values to specify a topic.

Note

The help button control is an advanced control that should only be used by Windows programmers and Windows help file authors. For more information about creating and compiling Windows help files and context-sensitive help, consult the Windows SDK or the Windows Help Author's Guide.

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

the following controls may later be added to Corel SCRIPT

SLIDER

Adds a slider to a dialog box.

Syntax
SLIDER Left%, Top%, Width%, Height%, Text$, Identifier

Syntax	Definition
Left%	Distance in dialog units from the inside of the dialog box's left border to the left side of the slider.
Top%	Distance in dialog units from the bottom of the dialog box's title bar to the top of the slider.
Width%	Slider width in dialog units.
Height%	Slider height in dialog units.
Text$	Label displayed to the right of the check box. Placing an ampersand (&) before a character provides a keyboard shortcut to selecting the check box.
Identifier	**Identifier** is the name of the check box control. It is also a variable that holds the return value that corresponds to the state of the check box. Optionally, you can use **Identifier** to set the default state of the check box.

Returns and Defaults	Condition
an integer	Check box is enabled, and displays a check mark.

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

HSCROLLBAR

Adds a horizontal scroll bar to a dynamic dialog box.

Syntax

HSCROLLBAR Left%, Top%, Width%, Height%, Text$, Identifier

Syntax	Definition
Left%	Distance in dialog units from the inside of the dialog box's left border to the left side of the horizontal scroll bar.
Top%	Distance in dialog units from the bottom of the dialog box's title bar to the top of the horizontal scroll bar.
Width%	Horizontal scroll bar width in dialog units.
Height%	Horizontal scroll bar height in dialog units.
Text$	Label displayed to the right of the check box. Placing an ampersand (&) before a character provides a keyboard shortcut to selecting the check box.
Identifier	**Identifier** is the name of the check box control. It is also a variable that holds the return value that corresponds to the state of the check box. Optionally, you can use **Identifier** to set the default state of the check box.

Returns and Defaults	Condition
an integer	Check box is enabled, and displays a check mark.

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

VSCROLLBAR

Adds a vertical scroll bar to a dynamic dialog box.

Syntax

VSCROLLBAR Left%**, Top**%**, Width**%**, Height**%**, Text$, Identifier**

Syntax	Definition
Left%	Distance in dialog units from the inside of the dialog box's left border to the left side of the vertical scroll bar.
Top%	Distance in dialog units from the bottom of the dialog box's title bar to the top of the vertical scroll bar.
Width%	Vertical scroll bar width in dialog units.
Height%	Vertical scroll bar height in dialog units.
Text$	Label displayed to the right of the check box. Placing an ampersand (&) before a character provides a keyboard shortcut to selecting the check box.
Identifier	**Identifier** is the name of the check box control. It is also a variable that holds the return value that corresponds to the state of the check box. Optionally, you can use **Identifier** to set the default state of the check box.

Returns and Defaults	Condition
an integer	Check box is enabled, and displays a check mark.

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

PROGRESS

Adds a progress indicator to a dynamic dialog box.

Syntax

CHECKBOX Left%**, Top**%**, Width**%**, Height**%**, Text$, Identifier**

Syntax	Definition
Left%	Distance in dialog units from the inside of the dialog box's left border to the left side of the progress indicator.
Top%	Distance in dialog units from the bottom of the dialog box's title bar to the top of the progress indicator.
Width%	Progress indicator width in dialog units.
Height%	Progress indicator height in dialog units.
Text$	Label displayed to the right of the check box. Placing an ampersand (&) before a character provides a keyboard shortcut to selecting the check box.
Identifier	**Identifier** is the name of the check box control. It is also a variable that holds the return value that corresponds to the state of the check box. Optionally, you can use **Identifier** to set the default state of the check box.

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

Using operating system and file commands

You can use Corel SCRIPT statements and functions to create a connection between your scripts and your Windows operating system. These commands allow you to do things such as:

- read from and write into Windows text files
- set the current folder
- create file lists
- create and delete files and folders

Note

- You can have up to ten Windows text files open at one time in your computer's memory. These files are assigned numbers from 1 to 10 when they are opened. Any time you refer to a file after it is opened, you use the number of the file rather than the filename.

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

GETAPPHANDLE

Returns the H-Instance for the application that is running the script. For example, if running the script from the Corel SCRIPT Editor, GETAPPHANDLE returns the Editor's application H-Instance. This function is used in conjunction with DLL calls that require the application's handle.

Syntax
hand{&} **= GETAPPHANDLE()**

Syntax	Definition
hand{&}	A long variable that is assigned the application's handle.

Example

```
hand = GETAPPHANDLE()
```

{button ,AL(`declare_lib;open_output;open_append;open_input;getwinhandle;getapphandle;',0,"Defaultoverview",)}
Related Topics

GETWINHANDLE

Returns the window handle for the window that is running the script. For example, if running the script from the Corel SCRIPT Editor, GETWINHANDLE returns the Editor's Windows handle. This function is used in conjunction with DLL calls that require the window's handle.

Syntax
hand{&} **= GETWINHANDLE()**

Syntax	Definition
hand{&}	A long variable that is assigned the window's handle.

Example

```
hand = GETWINHANDLE()
```

{button ,AL(`declare_lib;open_output;open_append;open_input;getwinhandle;getapphandle;',0,"Defaultoverview",)}
Related Topics

CLOSE

Closes a Windows text file opened with an OPEN statement (OPEN...APPEND or OPEN...OUTPUT).

Syntax
CLOSE #num% ,...

Syntax	Definition
num%	File number of the file to be closed; a constant or an integer variable with the value 1-10, inclusive. If not specified, all open files are closed. The # sign is optional.

Note

- In your scripts, every OPEN command should have a corresponding CLOSE statement.

Example

```
CLOSE
```

Closes all open files

```
CLOSE #1
```

Closes the file opened as 1.

```
CLOSE 1
```

Closes the file opened as 1.

```
CLOSE 1, 3, 5
```

Closes the files opened as 1, 3, and 5.

{button ,AL(`open_append;open_input;open_output;;;',0,"Defaultoverview",)} Related Topics

EOF

Returns TRUE if the file pointer is at the end of an open Windows text file in your computer's memory. Returns FALSE if the file contains data beyond the pointer. The statement is often used to determine whether to continue processing a file.

Syntax

EOF (#**num%)**

Syntax	Definition
#**num%**	File number of the open file to be examined; a constant or an integer variable with the value 1-10, inclusive. The # sign is optional.

Example

```
IF EOF(1) THEN CLOSE 1
```

If the pointer is at the end of the file, then close the file.

{button ,AL(`lof;seek;open_append;print;write;',0,"Defaultoverview",)} **Related Topics**

FILEMODE

Returns the file mode of an open Windows text file in your computer's memory.

Syntax

FILEMODE(num%)

Syntax	Definition
num%	File number of the file to be examined; a constant or an integer variable with the value 1-10, inclusive.

Returns

The function returns one of the following:

- 1 if file num% was opened for input
- 2 if file num% was opened for output
- 8 if file num% was opened for append

Note

- This the only Corel SCRIPT file function that doesn't have an optional # sign in front of the file number.

Example

```
i% = FILEMODE(1)
```

If file 1 was opened for input, then **i%** is set to 1. If file 1 was opened for output, then **i%** is set to 2. If file 1 was opened for append, then **i%** is set to 8.

```
OPEN "C:\example.txt" FOR APPEND AS 2
i% = FILEMODE(2)
```

Assigns 8 to the variable **i%**.

{button ,AL(`OPEN_APPEND;OPEN_INPUT;OPEN_OUTPUT;;;',0,"Defaultoverview",)} Related Topics

FILEPOS

Returns the current file position of the file pointer for the specified file.

Syntax
FILEPOS (#num%)

Syntax	Definition
#**num%**	File number of the file to be examined; a constant or an integer variable with the value 1-10, inclusive. The # sign is optional.

Note

- The seek always starts from the first position in the file.

Example

```
OPEN "C:\HELLO.TXT" FOR INPUT AS 2
SEEK 2, 12
position% = FILEPOS(2)
```

Assigns 12 to the variable **i%**.

{button ,AL(`lof;seek;open_append;print;write;',0,"Defaultoverview",)} **Related Topics**

FREEFILE

Returns the lowest file number not associated with an open Windows text file in the computer's memory.

Syntax
FREEFILE

Example

```
OPEN "temp.out" FOR OUTPUT AS 1
OPEN "temp2.out" FOR OUTPUT AS 5
i% = FREEFILE
```

The lowest available file number in this example is 2 because 1 is already being used, so **i%** is set to 2.

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

FINDFIRSTFOLDER, FINDNEXTFOLDER

Use the FINDFIRSTFOLDER and FINDNEXTFOLDER functions to assemble or perform an operation on a list of files, folders, or both. The FINDFIRSTFOLDER function is used to locate the first file or folder in a folder that meets a specified search criteria. The FINDNEXTFOLDER function is used to locate the next file or folder, and so on, that meets the specified search criteria set by the FINDFIRSTFOLDER. The FINDNEXTFOLDER function must be used in conjunction with the FINDFIRSTFOLDER function.

Syntax

FolderFileName$ = FINDFIRSTFOLDER(searchcriteria, attributes)
FolderFileName$ = FINDNEXTFOLDER( )

Syntax	Definition
searchcriteria	Specifies the files or folders to search for. You can include wild-card characters (* or ?).
attributes	The type of files or folders you want to use. You can use the OR operator to specify types (default value is all attributes): 0 = read-only files and folders 1 = read-only files and folders 2 = hidden files and folders 4 = system files and folders 16 = folder and folders 32 = archive files and folders 128 = normal files and folders 256 = temporary files and folders 2048 = compressed files and folders

Example

```
DIM bmpfile(100) AS STRING   'really large array (<= 100 files)
i% = 1                        'counter variable
bmpfile(i) = "C:\myimages\" + FINDFIRSTFOLDER("C:\myimages\*.bmp", 1 OR 2 OR 4 OR 16 OR 32)
DO WHILE bmpfile(i) <> "C:\myimages\"
    i = i + 1                 'counter increment
    bmpfile(i) = "C:\myimages\" + FINDNEXTFOLDER()
LOOP
```

In the above example, a list of bitmap files in the **C:\myimages** folder is created by assigning the bitmap file names to the **bmpfile** array.

```
CURRFOLDER = "C:\myimages"
bmpfile$ = FINDFIRSTFOLDER("*.bmp", 1 OR 2 OR 4 OR 16 OR 32)
DO WHILE bmpfile <> ""
    KILL bmpfile
    bmpfile = FINDNEXTFOLDER()
LOOP
```

In the above example, bitmap files in the **C:\myimages** folder are deleted using the **KILL** statment.

{button ,AL(`CURRFOLDER;filemode;getfileattr;getcurrfolder;setcurrfolder;',0,"Defaultoverview",)} **Related Topics**

FILEATTR

Use the FILEATTR function to return a file's attributes.

Syntax
retval{&} **= FILEATTR(FileName$)**

Syntax	Definition
retval{&}	Returns the attributes of **FileName$**: 0 = file doesn't exist 1 = read-only files or folder 2 = hidden files or folder 4 = system files or folder used by operating system 16 = folder 32 = archive files or folder 128 = normal files (no attributes set) 256 = temporary files or folder
FileName$	Specifies the file for which the attributes are returned.

Example

```
retval = FILEATTR("C:\myfiles\mysetup.txt")
```

If **mysetup.txt** was a read-only, hidden, and a system file, **retval** equals 7.

In cases which multiple attributes are returned, you can use the AND (bitwise) to determine specific attributes. To determine if **mysetup.txt** was a read-only file you could use the following syntax:

```
IF 1 AND retval THEN readOnly$ = "Yes" ELSE readOnly$ = "No"
```

1 is the read-only attribute. The variable **readOnly** is assigned a string based on bitwise comparison. In this case **readOnly** is assigned "Yes".

{button ,AL(`CURRFOLDER;filemode;getfileattr;FINDFIRSTFOLDER_FINDNEXTFOLDER;setcurrfolder;',0,"Defaultoverview",)} Related Topics

FILEDATE

Use the FILEDATE function to return a file's last modification date.

Syntax
retval = FILEDATE(FileName$)

Syntax	Definition
retval	Assigned the date of the specified file's last modification as date data type. If the file is not found, 0 is returned.
FileName$	Specifies the file for which the date property is returned.

Note

- If the file is not closed when the function is executed, the function returns 12:00 AM.

Example

```
retval = FILEDATE("C:\myfiles\mytext.txt")
```

{button ,AL(`CURRFOLDER;filemode;getfileattr;FINDFIRSTFOLDER_FINDNEXTFOLDER;setcurrfolder;',0,"Defaultoverview",)} Related Topics

FILESIZE

Use the FILESIZE function to return a file's size in bytes.

Syntax
retval = FILESIZE(FileName$)

Syntax	Definition
retval	Assigned the size of the specified file in bytes. If the file is not found, 0 is returned.
FileName$	Specifies the file for which the date property is returned.

Example

```
retval = FILESIZE("C:\myfiles\mytext.txt")
```

{button ,AL(`CURRFOLDER;filemode;getfileattr;FINDFIRSTFOLDER_FINDNEXTFOLDER;setcurrfolder;',0,"Defaultoverview",)} Related Topics

INPUT

Starting at the pointer, reads a number of bytes (characters) from a Windows text file.

Syntax
INPUT(bytes{%|&|!|#|@|$}**,** #**num**{%|&|!|#|@|$}**)**

Syntax	Definition
bytes{%|&|!|#|@|$}	The number of bytes to be read.
#**num**{%|&|!|#|@|$}	File number of the file to be examined with the value 1-10, inclusive. The # sign is optional.

Example

```
myExtract$ = INPUT(50, #1)
```

Reads 50 characters from file 1 and assigns them to the variable **myExtract$**.

{button ,AL(`INPUT;LINE_INPUT;SEEK;FILEPOS;EOF;LOF;',0,"Defaultoverview",`main')} Related Topics

INPUT

Reads from a file to a list of variables. Values in the file are separated by commas. Character strings that include commas must be enclosed in quotation marks. The quotation marks do not appear in the variable.

Syntax

**INPUT #num%, var{%|&|!|#|@|$}**, var{%|&|!|#|@|$}, ...

Syntax	Definition
#num%	File number of the file to be examined with the value 1-10, inclusive.
var{%|&|!|#|@|$}	The variables that receive the fields as they are read from the file.

Note

- The number sign (#) is required in the syntax.
- Numeric data with decimals can only be inputted if the period (**.**) is used as the decimal separator.
- Dates can only be inputted if in the standard U.S. format (M/d/yy h:mm:ss TT).
- Booleans can only be inputted as TRUE or FALSE.

Example

```
INPUT #1, title$, number%
```

Reads a string and a numeric variable from file 1. Assigns the values to the string variable **title$** and the integer variable **number%**. The contents of file 1 could be in either of two formats:

```
"Ottawa", 50
"Huckleberry Finn", 101
"Hamlet", 220
```

or

```
"Ottawa", 50, "Huckleberry Finn", 101, "Hamlet", 220
```

{button ,AL(`input_dollar;open_input;line_input;write;;',0,"Defaultoverview",)} **Related Topics**

KILL

Deletes a file. This statement is the same as clicking File, Delete in the Windows Explorer or in My Computer. You can only use KILL to delete files in the current folder.

Syntax
KILL fileName$

Syntax	Definition
fileName$	Name of the file to be deleted. You can use wild cards (* and ?) if you want to delete a group of files. For example, **script*.*** deletes all the files in the current folder beginning with **script**. Using **script?.*** deletes all the files in the current folder that begin with **script** and are followed by only one more character.

Example

```
KILL "temp.out"
```

Deletes the file "temp.out" in the current folder.

Note

- Use the CURRFOLDER to set the current folder where files will be deleted.

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

LINE INPUT

Reads the next line from an open Windows text file in your computer's memory into a string.

Syntax

LINE INPUT #num%, string$

Syntax	Definition
#num%	File number of the file to be examined; a constant or an integer variable with the value 1-10, inclusive.
string$	The string variable to hold the line from the file.

Note

- The number sign (#) is required in the syntax.

Example

```
LINE INPUT #1, MyString$
```

Reads the next line from file 1 and places the string in the variable **MyString$.**

{button ,AL(`input_dollar;input;open_input;line_input;;;',0,"Defaultoverview",)} **Related Topics**

LOF

LOF (Length of File) returns the number of bytes in an open Windows text file .

Syntax
LOF(#num%)

Syntax	Definition
#**num%**	File number of the file to be examined; a constant or an integer variable with the value 1-10, inclusive. The # sign is optional.

Example

```
bytes% = LOF(1)
```

Sets bytes% to the number of bytes in file 1.

{button ,AL(`eof;seek;open_append;print;write;',0,"Defaultoverview",)} **Related Topics**

MKFOLDER

Creates a new folder. You can also use MKFOLDER as a function: it returns -1 if the folder was created, 0 if not.

Syntax
MKFOLDER folderName$

Syntax	Definition
folderName$	Name of the folder to be created.

Example

```
MKFOLDER "work"
```

Creates the folder **work** as a subfolder of the current folder.

```
success = MKFOLDER "work"
```

Creates the folder **work** as a subfolder of the current folder and assigns -1 to **success%**.

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

OPEN...APPEND

Opens a Windows text file for adding or appending data to the end of the file. Data is appended using the PRINT or WRITE statement. The FOR clause is required.

Syntax

OPEN fileName$ FOR APPEND AS #num%

Syntax	Definition
fileName$	Name of the file to open. If the file doesn't exist, it is created and then opened.
#num%	Number (1 through 10) associated with the file; used to access the file. The # sign is optional.

Example

```
OPEN "oldfile" FOR APPEND AS 3
```

Opens the file **oldfile** for the appending of sequential data.

{button ,AL(`input_dollar;open_append;open_input;open_output;ACCESS;CLOSE;EOF;FILEMODE;FILEPOS;freefile;input;line_input;lof;print;seek;write',0,"Defaultoverview",`main')} Related Topics

OPEN...INPUT

Opens a Windows text file so that data can be read from it using the INPUT or LINE INPUT statement.

Syntax
OPEN fileName$ FOR INPUT AS #num%

Syntax	Definition
fileName$	Name of the file to open. An error occurs if the file doesn't exist.
#**num%**	Number (1 through 10) associated with the file; used to access the file. The # sign is optional.

Example

```
OPEN string$ FOR INPUT AS 2
```

Opens the file specified by **string$** for input.

{button ,AL(`input_dollar;open_append;open_input;open_output;ACCESS;CLOSE;EOF;FILEMODE;FILEPOS;freefile;input;line_input;lof;print;seek;write',0,"Defaultoverview",`main')} Related Topics

OPEN...OUTPUT

Opens a Windows text file so that data can be written into it using the PRINT or WRITE statement. The FOR clause is required.

Syntax

OPEN fileName$ FOR OUTPUT AS #num%

Syntax	Definition
fileName$	Name of the file to open. If the file doesn't exist, it is created and then opened. If the file does exist, its contents are deleted.
#**num%**	Number (1 through 10) associated with the file; used to access the file. The # sign is optional.

Example

```
OPEN "c:\temp\workfile.tmp" FOR OUTPUT AS 1
```

Opens the file **c:\temp\workfile.tmp** for output.

{button ,AL(`input_dollar;open_append;open_input;open_output;ACCESS;CLOSE;EOF;FILEMODE;FILEPOS;freefile;input;line_input;lof;print;seek;write',0,"Defaultoverview",`main')} Related Topics

PRINT

Prints expression(s) to an open Windows text file in your computer's memory. Character strings are not enclosed in quotes when written out. Numeric expressions have either a leading space or a negative sign in addition to a trailing blank space. Each expression follows immediately after the previous one.

Syntax

PRINT #num%, expression1, expression2 , ...

Syntax	Definition
#num%	File number of the file to be examined; a constant or an integer variable with the value 1-10, inclusive.
expression1, expression2	String or numeric expressions to be written to the file.

Note

- The number sign (#) is required in the syntax.
- Numeric data is printed using the period (**.**) as decimal separator.
- Dates are printed in the standard U.S. format (M/d/yy h:mm:ss TT).
- Booleans are printed as TRUE or FALSE.

Example

```
PRINT #1, "HEADING: ", string1$, ",", my_int%
```

Outputs "HEADING: " followed by the value of **string1$**, a comma, and the value in **my_int%**. No spaces are added between the four items, except for those contained in the literal string.

{button ,AL(`seek;print;write;open_append;open_output;input;;',0,"Defaultoverview",)} Related Topics

RMFOLDER

Removes an existing folder. The folder must be empty before it can be deleted. You can also use RMFOLDER as a function: it assigns -1 if the folder was removed, 0 if not.

Syntax
RMFOLDER folderName$

Syntax	Definition
folderName$	The name of the folder to remove.

Example

```
RMFOLDER "C:\TEMP\WORK"
```

Removes the WORK folder from the \TEMP folder.

```
success% = RMFOLDER "C:\TEMP\WORK"
```

Removes the WORK folder from the \TEMP folder and assigns -1 to **success%**.

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

SEEK

To prepare for subsequent input, moves the file pointer to a specific byte position in an open Windows text file in your computer's memory. Characters, spaces, tabs, and line breaks, are counted as characters.

Syntax

SEEK #**num%, position%**

Syntax	Definition
num%	File number of the file to be examined; a constant or an integer variable with the value 1-10, inclusive. The # sign is optional.
position%	The byte position to SEEK (1 is 1st byte).

Note

- The seek always starts from the first position in the file.
- The seek range is limited from 1 to length of the file (LOF) + 1

Example

```
SEEK 1, 100
```

Moves the file pointer to the 100th byte in file 1.

{button ,AL(`OPEN_APPEND;OPEN_INPUT;OPEN_OUTPUT;lof;;',0,"Defaultoverview",)} Related Topics

WRITE

Writes expressions to an open Windows text file in your computer's memory. Character strings are enclosed in quotes when written out. Each expression is separated by a comma.

Syntax

WRITE **#num%, expression1** , expression2, ...

Syntax	Definition
#num%	File number of the file to be examined; a constant or an integer variable with the value 1-10, inclusive.
expression1, expression2	The expressions to write to the file.

Note

- The number sign (#) is required in the syntax.
- Numeric data is written using the period (**.**) as decimal separator.
- Dates are written in the standard U.S. format (M/d/yy h:mm:ss TT).
- Booleans are written as TRUE or FALSE.

Example

```
MyString$ = "some text"
MyInt% = 55
WRITE #1, MyString$, "Literal Text", MyInt%
```

Writes the following to file 1:
"some text", "Literal Text", 55

{button ,AL(`seek;print;write;open_append;open_output;input;;',0,"Defaultoverview",)} **Related Topics**

CURRFOLDER

The CURRFOLDER statement is used to assign the current folder name to a variable, or to change the current folder.

Syntax to change the current folder
CURRFOLDER = folderName$

Syntax to assign the current folder name
folderName$ = CURRFOLDER

Syntax	Definition
folderName$	The name of the folder including drive.

Example

```
folderName$ = CURRFOLDER
CURRFOLDER = "c:\corel60\work"
OPEN "HAPPY.TXT" FOR INPUT AS 2
CURRFOLDER = folderName$
```

The above example, assigns the current folder to the variable **folderName$**. The current folder is then set to the work folder, and the HAPPY.TXT file in the work folder is opened for input. The last statement resets the active folder to **folderName$**, the original active folder.

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

COPY

The COPY command copies a file.

You can also use COPY as a function: it returns -1 (TRUE) if the COPY operation is successful, 0 (FALSE) if not.

Syntax
COPY file1$, file2$, overwrite**%**

Syntax	Definition
file1$	Name of the file to copy. **file1$** can include drive and folder.
file2$	Name of the file where **file1$** is to be copied. **file2$** can include drive and folder.
overwrite%	If **file2$** already exists, determines whether to overwrite the existing file: 0 = copy and overwrite (default if omitted) 1 = overwrite fails

Example

```
DIM x AS STRING
DIM y AS STRING
x = "C:\work\example1.cdr"
y = "D:\work\example1.cdr"
success = COPY x, y, 0
```

The above example copies the **example1.cdr** file to the work folder on the D drive, and assigns -1 to **success%**.

{button ,AL(`FINDFIRSTFOLDER_FINDNEXTFOLDER;rename;copy;CURRFOLDER;rmfolder;currfolder;;',0,"Defaultoverview",)} Related Topics

RENAME

The RENAME command changes the name of a file or folder, or can be used to move a file. You cannot move a folder using the RENAME command.

You can also use RENAME as a function: it returns -1 (TRUE) if the RENAME operation is successful, 0 (FALSE) if not.

Syntax

RENAME file_folder1$, file_folder2$, overwrite**%**

Syntax	Definition
file_folder1$	Name of the file or folder to move. **file_folder1$** can include drive and folder path specifics ▪ if path specifics are not included, RENAME assumes the current folder.

File_folder2$** Name of the file where **file_folder1$ is to be moved. **file_folder2$** can include drive and folder path specifics {emdash.bmp} if path specifics are not included, RENAME assumes the current folder.

overwrite% If **file_folder2$** already exists, determines whether to overwrite the existing file (you cannot overwrite existing folders):

0 = rename and overwrite

1 = overwrite fails (default if omitted)

Example

```
DIM x AS STRING
DIM y AS STRING
x = "C:\work\example1.cdr"
y = "D:\work\example1.cdr"
RENAME x, y, 0
```

The above example moves the **example1.cdr** file to the work folder on the D drive.

```
DIM x AS STRING
DIM y AS STRING
x = "C:\work\example1.cdr"
y = "C:\work\example2.cdr"
success% = RENAME x, y, 0
```

The above example renames the **example1.cdr** file to **example2.cdr**, and assigns -1 to **success%**.

{button ,AL(`FINDFIRSTFOLDER_FINDNEXTFOLDER;rename;copy;CURRFOLDER;rmfolder;currfolder;;',0,"Defaultoverview",)} Related Topics

Using flow control statements

Corel SCRIPT scripts generally execute in a linear manner; that is, in the order in which they appear. Flow control statements allow you to dictate how a script can be executed. Additionally, using flow controls can make your scripts more flexible and efficient.

You can use flow control statements to:

- Execute specific statements only if an expression meets a condition of TRUE or FALSE.
- Execute different sets of statements depending on the value of a variable.
- Repeat statements while or until an expression meets a condition of TRUE or FALSE.
- Repeat statements a specific number of times.
- Call a subroutine or function.
- Go to a different line in the script.
- Terminate a script's execution.
- Direct script execution to a Corel application.

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

CALL

Executes a named subroutine but you must first DECLARE all subroutines before you can CALL them.

You are not required to use the **CALL** keyword when calling a subroutine. If arguments are included, you must enclose them with parentheses if the word CALL is used. An empty argument list indicated by ( ) is not allowed.

Syntax

CALL name (parameter1, parameter2, ...)
name (parameter1, parameter2, ...)

Syntax	Definition
name	Name of the subroutine to call; not case-sensitive.
parameter	One or more optional arguments (variables, constants, or expressions), separated by commas, that are passed to the subroutine.

{button ,AL(`DECLARE;SUB;;;;',0,"Defaultoverview",`main')} Related Topics

DO...LOOP

Repeats statements while a condition is true or until it becomes true. Note that the first form of syntax may never execute statements if an expression is or is not true, but that the second form always executes statements at least once.

Syntax

DO {WHILE | UNTIL} expression
[statements]
LOOP

or

DO
[statements]
LOOP {WHILE | UNTIL} expression

Syntax	Definition
expression	An expression that is evaluated to determine whether to continue the loop.
[statements]	Series of script instructions to execute and repeat.

Note

- By placing the condition at the end of the loop, the loop will always be executed once before the condition is tested.
- You can nest DO...LOOP statements inside each other up to 20 times.

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

Examples for DO...LOOP

```
i% = 5
DO WHILE i% > 0
    i% = i% - 1
LOOP
```

The variable **i%** starts with a value of 5. The loop continues for as long as **i%** has a value greater than 0. Once **i%** equals 0 and the condition (**DO WHILE i% > 0**) is again tested, processing continues at the next statement after the LOOP statement. In this example, the loop executes five times. If the condition were changed to "**i% = 0**", the loop would never execute because the condition would be false immediately.

```
i% = 5
DO UNTIL i% = 0
    i% = i% - 1
LOOP
```

This example is functionally identical to the previous example. The loop executes five times before the condition is true. If the condition were changed to "**i% > 0**", the loop would never execute.

```
i% = 5
DO
    i% = i% - 1
LOOP UNTIL i% = 0
```

By placing the condition at the end of the loop, the loop will always be executed once before the condition is tested. However, in the following example, the loop will still execute five times.

```
i% = 5
DO
    i% = i% - 1
LOOP WHILE i% > 0
```

Again, the loop will execute five times before the condition is no longer true. If the condition were changed to "**i% = 0**" the loop would execute only once before the condition is tested.

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

END

Ends a SELECT CASE, FUNCTION, SUB, WITHOBJECT or DIALOG construct.

Syntax

END {SELECT | FUNCTION | SUB | WITHOBJECT | DIALOG}
You must specify the type of subgroup being ended.

Example

```
SELECT CASE i%
    CASE 0
        REM Case statements go here.
END SELECT

FUNCTION UserFunction%( )
    REM Function processing statements go here.
END FUNCTION

SUB Subroutine( )
    REM Subroutine statements go here.
END SUB

WITHOBJECT "CorelDraw.Automation.6"
    REM CorelDRAW commands and functions go here.
END WITHOBJECT

BEGIN DIALOG Dialogbox1 120, 45, "Sample Dialog box"
    REM Dialog box statements go here.
END DIALOG
```

The script compiler must match a SELECT CASE, FUNCTION, SUB, WITHOBJECT or BEGIN DIALOG statement with a corresponding END statement, as shown in the examples above.

{button ,AL(`WITH_Corel_Application;SELECT_CASE;BEGIN_END_DIALOG;SUB_END_SUB;FUNCTION_END_FUNCTION;;',0,"Defaultoverview",)} Related Topics

FOR...NEXT

Repeats (or loops) a group of instructions a specified number of times. The variable loop **counter** has an initial value of **start** and is changed each time the loop statements are executed. The **counter** changes by the value set in the optional **increment** parameter. If the **increment** parameter is not used, **counter** steps, or is increased, by one each time the loop statements are executed.

Syntax

FOR counter{%|&|!|#|@} **= start**{%|&|!|#|@} **TO end**{%|&|!|#|@} STEP increment{%|&|!|#|@}
[statements]
NEXT counter{%|&|!|#|@}

Syntax	Definition
counter{%|&|!|#|@}	Numeric variable used as counter. Cannot be an array element.
start{%|&|!|#|@}	Initial value of the counter.
TO	Used to separate the **start** and **end** parameters.
end{%|&|!|#|@}	Final value of the counter.
STEP	Optional syntax used with the increment parameter.
increment{%|&|!|#|@}	Amount the counter is incremented each time through loop. If omitted, the default = 1. Can be a variable or a constant.
[statements]	Series of script instructions to execute and repeat.

Note

- The **increment** parameter can be a positive or negative value. If the **increment** parameter is a positive value, the loop continues to execute as long as **counter** is less than or equal to **end**. If the **increment** parameter is a negative value, the loop continues to execute as long as **counter** is greater than or equal to **end**.
- If set the **start** or **end** parameter is outside the **counter's** data type range, an endless loop can occur. For example, if **counter** is an integer and **end** is set to 33000, an endless loop will occur since 33000 beyond the range of an integer. For more information about data types, see Corel SCRIPT data type summary.
- You can nest FOR...NEXT statements inside each other up to 20 times.

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

Examples for FOR...NEXT

```
FOR i% = 1 TO 10 STEP 2
    intArray%(i%) = i%
NEXT i%
```

Every other element of the array intArray% (elements 1, 3, 5, 7, and 9) are given the value of i%.

```
FOR i% = 1 TO UBOUND(stringArray$)
    stringArray$(i%) = "string"
NEXT i%
```

All the elements of the array stringArray$ are assigned the value "string".

Note

- If you nest FOR...NEXTs within FOR...NEXTs, you should give each a **counter%** a unique name as shown in the following example:

```
For a% = 1 To 4
    [statements]
    For b% = 1 To 4
        [statements]
        For c% = 1 To 4
            [statements]
        Next c%
    Next b%
Next a%
```

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

GOTO

Directs program execution to the line specified by **linelabel**.

A GOTO statement can branch only to another statement at the same procedural level in the script. For example, you can only use a GOTO statement in the main section of the script to go to another line in the main section. You cannot use GOTO to enter or exit a subroutine or a function.

Syntax

GOTO linelabel

Syntax	Definition
linelabel	Label of the line to go to (do not include the colon in the GOTO statement)

Note

- In the script, the line label cannot be preceded by spaces or tabs and must be followed by a colon. It cannot be more than 256 characters long.
- Using GOTO statements can make your script difficult to read and debug. You should provide remarks statements when using the GOTO statement. Consider using conditional or looping statements instead of the GOTO statement.

Example

```
REM Statements...
IF i% = 3 THEN GOTO BreakOut
REM Statements execute here if i% is not equal to 3
BreakOut:
REM More statements execute here.
```

If **i%** is equal to 3, then the statements between the GOTO and the label are skipped, and the processing starts with the first statement after the label. The statements after the label will execute even if **i%** is not equal to 3.

Note

- Using the GOTO statement can easily cause an endless loop in a program. The following example creates an endless loop:

```
REM Statements...
I% = 3
Breakout:
If I% = 3 then GOTO Breakout
```

The above example continously loops between the last two statements.

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

IF...THEN...ELSE...ENDIF

Using the IF statement allows for conditional execution of script instructions based on TRUE or FALSE conditions. If the condition is TRUE, the program performs the THEN statement; if the condition is FALSE, the program performs the ELSE or ELSEIF statement, if included.

If only one THEN statement is needed, and it is included on the same line as the IF condition, then no ENDIF is required. If the THEN statement is more than one line, then ENDIF is required.

Syntax

IF condition1 THEN [statement] ELSE [statement]

or

IF condition1 THEN
[statements]
ELSEIF condition2 THEN
[statements]
ELSE
[statements]
ENDIF

Syntax	Definition
condition1, condition2	A condition to evaluate. If the condition is TRUE, then the statements following the keyword THEN are executed. If the condition is FALSE, then the processing continues at the next ELSEIF statement, if present, or at the statement following the ELSE keyword, if present.
[statement]	Script instruction to execute.
[statements]	Series of script instructions to execute.

Note

You can nest IF statements inside each other up to 20 times.

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

Examples for IF...THEN...ELSE...ENDIF

```
IF i% = 0 THEN MESSAGE "The variable is 0."
```
If the variable **i%** has a value of 0, a message dialog box appears.

```
IF i% = 0 THEN MESSAGE "The variable is 0." ELSE BEEP
```
If the variable **i%** has a value of 0, a message appears, otherwise a beep is sounded. Because the entire IF statement is on one line, no ENDIF is needed.

```
IF i% = 0 THEN
    MESSAGE "The variable is 0"
    i% = 1
ELSEIF i% = 1 THEN
    MESSAGE "The variable is less than 2"
ELSEIF i% = 2 THEN
    MESSAGE "The variable is greater than 1"
ELSE
    BEEP
ENDIF
```

You must use the multi-line IF...THEN...ENDIF when there is a second condition to test after the first IF or when there is more than one statement to process as a result of the condition. Repeat the ELSEIF statements for as many conditions as needed. If many conditions must be tested, you might want to use the SELECT CASE statement. Note that in the example, even though i% is assigned a value of 1 in the i% = 0 condition, the ELSEIF i% = 1 condition does not execute. Once a condition is TRUE, processing continues after the ENDIF statement once the statements that meet the condition execute.

You can evaluate variables without assigning results. For example:

```
IF (abc<=15.3) then BEEP ELSE MESSAGE "It's greater than 15.3"
```

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

SELECT CASE...END SELECT

Compares the value of a test expression to the values in the CASE statements and executes blocks of statements dependent on their relationships. If none of the values in the CASE statements match the test expression, the CASE ELSE statements are executed.

Syntax

SELECT CASE testexpression
    CASE {caseexpression | caseexpression TO caseexpression | IS reloperator caseexpression} ,...
        [statements]
    CASE ELSE
        [statements]
END SELECT

Syntax	Definition
testexpression	A combination of numbers, strings, variables, constants, and operators that return a result.
caseexpression	An expression that evaluates to the same type as testexpression.
reloperator	A relational operator such as =, <>, >, <, >=, or <=.
[statements]	Program instructions conditionally executed; any number of statements on one or more lines.
TO	Corel SCRIPT keyword used to specify a range of values. The smaller value must precede TO.
IS	Corel SCRIPT keyword used with relational operators to specify a range of values.

{button ,AL(`IF_THEN_ELSE_ENDIF;Relational_Operators;;;;',0,"Defaultoverview",`main')} Related Topics

Example for SELECT CASE

```
SELECT CASE i%
    CASE -10 TO -5, 5 TO 9, IS = 10
         MESSAGE "Between -10 and -5 or 5 and 10"
    CASE -4 TO -1, 2 TO 4
         MESSAGE "Between -4 and 4 but not 0 or 1"
    CASE 0, 1
         MESSAGE "Zero or one"
    CASE ELSE
         MESSAGE "Greater than 10 or less than -10"
END SELECT
```

In the above example:

- If **i%** is between -10 and -5 or 10 and 5, the message "Between -10 and -5 or 10 and 5" appears.
- If **i%** is between -4 and 4, but is not zero or one, the message "Between -4 and 4 but not 0 or 1" appears.
- If i% is zero or 1, the message "Zero or one" appears.
- If **i%** is not any of those numbers, the message "Greater than 10 or less than -10" appears.

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

STOP

Stops execution of a running Corel SCRIPT script.

Syntax
STOP

Example

```
IF i% = 3 THEN STOP
```

If the value of i% is 3, then the script stops immediately. No other statements are executed.

The STOP statement is often used to stop a running script when a Cancel button is pressed in a dialog box as shown in the following example:

```
BEGIN DIALOG Buttons1 55, 34, 236, 40, "BUTTON example"
    OKBUTTON 21, 12, 40, 14
    CANCELBUTTON 71, 12, 40, 14
    PUSHBUTTON 121, 12, 40, 14, "&Push"
    HELPBUTTON 171, 12, 40, 14, "C:\Help.hlp", 1044
END DIALOG
ret = DIALOG(Buttons1)
IF ret = 1 THEN MESSAGE "OK button chosen"
IF ret = 2 THEN STOP    'Cancel pressed
IF ret = 3 THEN MESSAGE "Push button chosen"
```

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

WHILE...WEND

Continuously executes statements as long as the condition is true.

Syntax

WHILE condition
[statements]
WEND

Syntax	Definition
condition	Any expression that can be called true or false.
[statements]	Series of script instructions to execute.

Example

```
i% = 5
WHILE i% > 0
    i% = i% - 1
WEND
```

The loop executes for as long as the condition **i%** is greater than 0 is true. The loop will execute five times.

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

EXIT

The EXIT statement is used to exit a procedure or a loop.

Syntax

EXIT DO	exit from a DO loop
EXIT FOR	exit from a FOR loop
EXIT FUNCTION	exit from a function
EXIT SUB	exit from a subroutine

Example

```
FOR i% = 1 TO 5
    IF i = 3 THEN EXIT FOR
NEXT i
```

In the above example, the script execution exits the FOR loop when **i** equals 3, not 5.

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

WITHOBJECT...END WITHOBJECT

Use the WITHOBJECT statement to run a script that calls commands in a Corel application. You can also use the WITHOBJECT statement to call any application that is an OLE automation server.

Syntax

WITHOBJECT application{$}
    [statements]
END WITHOBJECT

or

WITHOBJECT application{$} **[statement]**

Syntax	Definition
application{$}	A string specifying the application to call. Not every Corel application supports Corel SCRIPT programming and script files. Click ▪ for a list of Corel applications that support Corel SCRIPT and application strings. This string is called the application's external name.

[statements] Series of script instructions to execute. Can be a combination of both Corel SCRIPT application commands and Corel SCRIPT intrinsic statements.

[statement]	Corel SCRIPT application command to execute.

Example

```
'Example 1
WITHOBJECT DRAW
    .InsertPages -1, 2
    .SetPaperColor 2, 0, 255, 0, 0
END WITHOBJECT


'Example 2
WITHOBJECT "CorelDraw.Automation.6"
    .InsertPages -1, 2
    .SetPaperColor 2, 0, 255, 0, 0
END WITHOBJECT


'Example 3
WITHOBJECT "CorelDraw.Automation.6" .InsertPages -1, 2
WITHOBJECT "CorelDraw.Automation.6" .SetPaperColor 2, 0, 255, 0, 0
```

The three above example all insert two pages into a CorelDRAW document and change the paper color..

Note

- You can also use the WITHOBJECT statement to call any application that supports OLE automation with an automation object. For example, you can call Microsoft Word 6.0 or Microsoft Excel 5.0 by using the external names "Word.Basic" or "Excel.sheet.5", respectively.
- The application the WITHOBJECT statement is calling does not have to be opened to be called.
- You can nest WITHOBJECT...END WITHOBJECT statements inside each other up to 20 times.
- If you have more than one session of your application open, the script is executed in the application which was opened first.
- Running a script with a combination of intrinsic statements and application commands from the Editor lets you take advantage of the Editor's testing and debugging features. However, once you are satisfied your script is running properly, you should run your script from the Corel application that uses the application command for significant time savings.

{button ,AL(`ole_cs;end;corel_script_programming_language;Executing_script_files;Corel_SCRIPT_advanced_programming_features;;',0,"Defaultoverview",)} Related Topics

REM

A non-executed remark, or comment, in a program. You can use an apostrophe (') instead of REM. Everything from REM (or the apostrophe) to the end of the line is a remark. The REM statement must be placed at the beginning of a line. The apostrophe can be placed anywhere in a line.

If a script's first line, second line, or both are REM statements, the comments are displayed in a Corel application's Run Script dialog box if the script is specified. The same two REM statements are also displayed in the status bar if the script is assigned to a tool bar button.

Syntax

REM This is a comment.

' This is also a comment.

Syntax	Definition
comment	any text to describe the script statements.

Example

```
REM Description: This is a description.
REM Created by: Your Name
' This first statement sets up the variables
DIM int2Array%(10) ' The rest of this line is a comment
' More processing follows here.
```

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

ON ERROR

The ON ERROR statement sets an error-handling routine. The error-handling routine is a series of instructions that are executed when an error occurs. If you don't use an error-handling routine, a run-time error stops script execution. For more information about run-time errors, see Script programming errors.

Note

- When you set ON ERROR, you are only setting an error handling routine within the procedure where it was set. A procedure can be the script's main section, a function, or a subroutine.
- When an error occurs, an error value is passed, or trapped, to the Corel SCRIPT global variable ERRNUM. For example, if a division by zero error occurs, ERRNUM equals 100. For more information, see Trappable Error Codes.

Syntax

ON ERROR { GOTO linelabel | RESUME NEXT | EXIT }

Argument	Definition
GOTO linelabel	Script execution is directed to the **linelabel** line when an error occurs. The **linelabel** must be in the same procedure where the error occurred. See GOTO for more information.
RESUME NEXT	Script execution continues to the line immediately following the line where the error occurred.
EXIT	Disables the ON ERROR setting in the current procedure.

Error-handling routines

An error-handling routine is not a separate procedure but a block of script instructions within the same procedure as the ON ERROR statement.

You can use the following syntax in the error-handling routine to return execution to a procedure from an error-handling routine:

Syntax to return execution	Definition
RESUME	Placing a RESUME statement at the end of an error-handling routine re-executes the script instruction which caused the error. If you use the RESUME statement, you should ensure that your error-handling routine resolves the error, otherwise an infinite loop is likely to occur.
RESUME NEXT	Placing a RESUME NEXT statement at the end of an error-handling routine directs script execution to the line immediately following the line where the error occurred.
RESUME AT **linelabel**	Placing a RESUME AT statement at the end of the error-handling routine directs script execution to the specified **linelabel** line. The **linelabel** line must be in the same procedure where the error occurred.

Note

- You should place a STOP, EXIT FUNCTION, or an EXIT SUB before an error-handling routine in a script to prevent it from being executed when no error has occurred.
- If an error occurs in an error-handling routine, script execution is redirected to the instruction that caused the error that initiated the error-handling routine.
- Not all Corel SCRIPT errors are trappable.

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

Example for ON ERROR and FAIL

```
DECLARE FUNCTION asknum$()
ON ERROR GOTO mainerrhan

num$=asknum$()
MESSAGE "You entered "+num$
STOP

' main error handling routine
mainerrhan:
    SELECT CASE errnum
    CASE 800
        MESSAGE "Please enter a higher number"
        RESUME
    CASE 801
        MESSAGE "Please enter a lower number"
        RESUME
    CASE ELSE
        MESSAGE "Unexpected error"
        RESUME NEXT
    END SELECT
STOP

' This function will ask for a number between 10 and 20.
FUNCTION asknum$
    ON ERROR GOTO asknumerror
    number$=INPUTBOX("Please enter a number between 10 and 20")
    somenum& = VAL(number$)
    IF somenum&<10 THEN FAIL 800
    IF somenum&>20 THEN FAIL 801
    asknum$=number$
    EXIT FUNCTION

    askNumError:
        ' Error handling is passed to the main section
END FUNCTION
```

In the above example, a function is used to create an input box that takes a number. If the number input is not between 10 and 20 inclusive, an error-handling routine is called. (The FAIL statement is used to simulate an error number.) The error-handling routine displays a message and then returns to the input box again.

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

FAIL

The FAIL statement simulates an error and sets the ERRNUM variable to the parameter value. FAIL can be used to force a call to an error-handling routine.

Syntax

FAIL errnum{ %|& }

Syntax	Definition
errnum{ %|& }	Any number, numeric variable or constant, or numeric expression. User defined error numbers are between 800 and 999, inclusive.

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

ABS

Calculates the absolute value of a real number. Absolute value is the positive value of a number.

Syntax
ABS(x{%|&|!|#|@}**)**

Argument	Definition
x{%|&|!|#|@}	Any number, numeric variable or constant, or numeric expression.

Examples

```
x% = ABS(-3)
y% = ABS(3)
```

Both the above examples return 3 to **x%** and **y%**.

{button ,AL(`abs;sgn;fix;int;;',0,"Defaultoverview",)} **Related Topics**

EXP

Raises *e* to a given exponent where *e* is the base of the natural logarithm which equals 2.718281828.

Syntax
EXP(x{%|&|!|#|@})

Argument	Definition
x{%|&|!|#|@}	Any positive number, numeric variable or constant, or numeric expression.

Note

- EXP is the inverse of the natural logarithm (LN).

Examples

```
x! = EXP(7.89)
```

In the above example, **x** equals 2670.444.

{button ,AL(`log;ln;exp;;;',0,"Defaultoverview",)} Related Topics

LOG

Calculates the base-10 logarithm of a number.

Syntax
LOG(x{%|&|!|#|@}**)**

Argument	Definition
x{%|&|!|#|@}	Any positive number, numeric variable or constant, or numeric expression, which represents the value.

Note

- The LOG function assumes a base of 10. If another base is needed, use LOG(x)/LOG(b) formula where **b** is the base.

Example

```
x = LOG(25)
y = LOG(5)
```

In the above example, **x** equals 1.397940 and **y** equals 0.6989700.

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

SIN

Calculates the sine of an angle measured in radians.

Syntax

SIN(x{%|&|!|#|@}**)**

Argument	Definition
x{%|&|!|#|@}	Any number, numeric variable or constant, or numeric expression. Specifies the angle measured in radians.

Note

- The result of SIN is between -1 to 1, inclusive.
- To convert degrees to radians, multiply degrees by 3.14159/180 (Pi is approximately equal to 3.14159) or use the ANGLECONVERT function.

Examples

```
degreeMeasure% = 45
MyResult! = SIN(3.14159/180*degreeMeasure%)
```

The above example returns the SIN of 45 degrees. The variable **MyResult!** equals 0.70710678.

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

TAN

Calculates the tangent of an angle in radians.

Syntax
TAN(x{%|&|!|#|@}**)**

Argument	Definition
x{%|&|!|#|@}	Any number, numeric variable or constant, or numeric expression. Specifies the angle measured in radians.

Note

- To convert degrees to radians, multiply degrees by 3.14159/180 (Pi is approximately equal to 3.14159) or use the ANGLECONVERT function.

Example

```
radianMeasure! = 0.5
MyResult! = TAN(radianMeasure!)
```

The above example returns the TAN of 0.5 radians. The variable **MyResult!** equals 0.546302.

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

COS

Calculates the cosine of an angle in radians.

Syntax

COS(x{%|&|!|#|@}**)**

Argument	Definition
x{%|&|!|#|@}	Any number, numeric variable or constant, or numeric expression. Specifies the angle measured in radians.

Notes

- The result of COS is between -1 and 1.
- To convert degrees to radians, multiply degrees by 3.14159/180 (Pi is approximately equal to 3.14159) or use the ANGLECONVERT function.

Examples

```
degreeMeasure% = 45
MyResult! = COS(3.14159/180*degreeMeasure%)
```

The above example returns the COS of 45 degrees as expressed in radians. The variable **MyResult!** equals 0.707106781.

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

ACOS

Calculates the inverse cosine (arccosine) of a value. The result is an angle measured in radians between 0 and p (where p is approximately 3.14152).

Syntax

ACOS(x{%|&|!|#|@}**)**

Argument	Definition
x{%|&|!|#|@}	Any number, numeric variable or constant, or numeric expression specifying the value to be calculated. Must be from -1 to 1, inclusive.

Note

- To convert the result from radians to degrees, use the ANGLECONVERT function or multiply the result by 180/3.14152.

Examples

```
v! = ACOS(-0.75)
w! = ACOS(0.75)
x! = ACOS(0)
y! = ACOS(1)
```

In the above example, **v!** is equal to 2.418858406, **w!** is equal to 0.722734248, **x!** is equal to 1.570796327, and **y!** is equal to 0.

{button ,AL(`Math_PASTE;ACOS;ASIN;ATAN;;',0,"Defaultoverview",)} Related Topics

ASIN

Calculates the inverse sine (arcsine) of a value. The result is an angle in radians bounded by -π /2 and π /2.

Syntax

ASIN(x{%|&|!|#|@}**)**

Argument	Definition
x{%|&|!|#|@}	Any number, numeric variable or constant, or numeric expression specifying the value to be calculated. Must be from -1 to 1, inclusive.

Note

- To convert the result from radians to degrees, use the ANGLECONVERT function or multiply the result by 180/3.14152.

Examples

```
w = ASIN(-0.75)
x = ASIN(0.75)
y = ASIN(0)
z = ASIN(1)
```

In the above example, **w!** is equal to -0.8480621, **x!** is equal to 0.8480621, **y!** is equal to 0, and **z!** is equal to 1.570796.

{button ,AL(`Math_PASTE;ACOS;ASIN;ATAN;;',0,"Defaultoverview",)} Related Topics

ATAN

Calculates the inverse tangent (arctangent) of a value. The result is an angle bounded by -π /2 (-90 degrees) and π /2 (90 degrees) measured in radians.

Syntax

ATAN(x{%|&|!|#|@}**)**

Argument	Definition
x{%|&|!|#|@}	Any number, numeric variable or constant, or numeric expression specifying the value to be calculated.

Note

- To convert the result from radians to degrees, use the ANGLECONVERT function or multiply the result by 180/3.14152.

Examples

```
w = ATAN(-0.75)
x = ATAN(0.75)
y = ATAN(0)
z = ATAN(1)
```

In the above example, **w!** is equal to -0.6435011, **x!** is equal to 0.6435011, **y!** is equal to 0, and **z!** is equal to 0.7853982.

{button ,AL(`Math_PASTE;ACOS;ASIN;ATAN;;',0,"Defaultoverview",)} **Related Topics**

DEC

Returns the conversion of a hexidecimal value into decimal notation.

Syntax
DEC(x{$}**)**

Argument	Definition
x{$}	A string representing a hexadecimal number.

Note

- Decimal notation is a numerical system based on groups of ten units.

Example

```
x! = DEC("A27")
y! = DEC("A27.89")
```

In the above example, **x** equals 2599 and **y** equals 2599.535.

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

HEX

Returns the conversion of a decimal value into hexadecimal notation.

Syntax
HEX(x{%|&|!|#|@}**)**

Argument	Definition
x{%|&|!|#|@}	Any number, numeric variable or constant, or numeric expression. Specifies

Note

- Hexadecimal notation is a numerical system based on groups of sixteen units.

Example

```
x$ = HEX(27)
y$ = HEX(27.25)
```

In the above example, **x** equals "1B" and **y** equals "1B.4".

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

FIX

Removes an argument's decimal or fraction and rounds towards 0. An integer is returned.

Syntax
FIX(x{%|&|!|#|@}**)**

Argument	Definition
x{%|&|!|#|@}	Any number, numeric variable or constant, or numeric expression .

Note

- Both INT and FIX return the integer portion of a given number. However, INT returns the greatest integer less than or equal to the number, while FIX returns the integer portion given, without any decimal points represented. As a result, -5.26 becomes -5 under the FIX function, and -6 under the INT function.

Example

```
vv = FIX(12.65)
yy = FIX(-47.29)
```

In the above example, **vv** equals 12 and **yy** equals -47.

```
v! = INT(12.65)
y! = INT(-47.29)
```

In the above example, **v** equals 12 and **y** equals -48.

{button ,AL(`abs;sgn;int;cint;;',0,"Defaultoverview",)} Related Topics

INT

Removes an argument's decimal or fraction and rounds down to the nearest integer. An integer is returned.

Syntax

INT(x{%|&|!|#|@}**)**

Argument	Definition
x{%|&|!|#|@}	Any umber, numeric variable or constant, or numeric expression.

Note

- Both INT and FIX return the integer portion of a given number. However, INT returns the greatest integer less than or equal to the number, while FIX returns the integer portion given, without any decimal points represented. As a result, -5.26 becomes -5 under the FIX function, and -6 under the INT function.

Examples

```
v! = INT(12.65)
y! = INT(-47.29)
```

In the above example, **v** equals 12 and **y** equals -48.

```
vv = FIX(12.65)
yy = FIX(-47.29)
```

In the above example, **vv** equals 12 and **yy** equals -47.

{button ,AL(`abs;sgn;fix;cint;;',0,"Defaultoverview",)} Related Topics

LN

Calculates the natural logarithm of a number.

Syntax
LN(x{%|&|!|#|@}**)**

Argument	Definition
x{%|&|!|#|@}	Any positive number, numeric variable or constant, or numeric expression.

Notes

- LN is the inverse of the EXP function, so LN(EXP(**x**)) equals **x**.
- Natural logarithms are based on the constant *e* which is equal to 2.718281828.

Examples

```
w = LN(1.2)
x = LN(30)
y = LN(1)
z = LN(EXP(30))
```

In the above example, **w** is equal to 0.1823215568, **x** is equal to 3.401197382, **y** is equal to 0, and **z** is equal to 30.

{button ,AL(`log;ln;exp;;;',0,"Defaultoverview",)} Related Topics

RANDOMIZE

Sets the random number generator seed to the integer portion of the argument value.

Syntax

RANDOMIZE x{%|&|!|#|@}

Argument	Definition
x{%|&|!|#|@}	Any number, numeric variable or constant, or numeric expression. This is an optional parameter.

Note

- Randomize uses the argument as a seed to start a new sequence of random numbers. Using RANDOMIZE without an argument initializes the seed to the system timer. If RANDOMIZE isn't used before calling RND, the same sequence of numbers will result every time the random number generator is used.

Examples

```
RANDOMIZE
RANDOMIZE 24
```

In the above example, the first line initializes the random number generator seed to the system timer. The second line uses 24 as the first seed in the random number generator's sequence.

The following example returns 5 random integers between 1 and 10 to the variable **a_random**. Each random value is also displayed in a message box.

```
RANDOMIZE
FOR x = 1 TO 5
    lower=1
    upper=10
    a_random = INT((upper - lower +1)*RND()+lower)
    MESSAGE a_random
NEXT x
```

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

RND

Returns a random number.

Syntax

RND(x{%|&|!|#|@}**)**

Argument	Definition
x{%|&|!|#|@}	Any number, numeric variable or constant, or numeric expression.

Note

- If RANDOMIZE isn't used before calling RND, the same sequence of numbers will result every time RND is used to create a random number.
- A positive number argument determines the upper bound of the area in which a random number can occur. The lower bound would be zero.
- A negative argument number determines the lower bound, with zero as the upper bound.
- If a seed is not provided, the system will generate a random number between zero and one.

Examples

```
w! = RND(0)
x! = RND()
y! = RND(-7)
z! = RND(24)
```

In the above example, **w!** returns the last generated number, **x!** returns a random number between zero and one, **y!** returns a random number between -7 and 0, and **z!** returns a random number between 0 and 24.

To create a random integer in a specified range, use the following formula:

```
INT((upper - lower +1)*RND()+lower)
```

where **upper** is the highest number in the specified range and **lower** is the lowest number in the specified range. The following example returns a random integer between 1 and 10 to the variable **a_random**.

```
lower = 1
upper = 10
a_random = INT((upper - lower +1)*RND()+lower)
```

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

SGN

Determines the sign (+ or -) of a number and returns -1 if the number is negative, 1 if the number is positive, and 0 if the number is 0.

Syntax

SGN(x{%|&|!|#|@}**)**

Argument	Definition
x{%|&|!|#|@}	Any number, numeric variable or constant, or numeric expression.

Examples

```
x% = SGN(5)
y% = SGN(0)
z% = SGN(-0.021)
```

In the above example, **x** is equal to 1, **y** is equal to 0, and **z** is equal to -1.

{button ,AL(`abs;sgn;fix;int;;',0,"Defaultoverview",)} **Related Topics**

SQR

Calculates the positive square root of a number.

Syntax
SQR(x{%|&|!|#|@}**)**

Argument	Definition
x{%|&|!|#|@}	Any non-negative number, numeric variable or constant, or numeric expression.

Examples

```
w = SQR(4)
x = SQR(0.175)
```

In this example, **w** is equal to 2 and **x** is equal to 0.4183300133.

{button ,AL(`abs;sgn;fix;int;;',0,"Defaultoverview",)} Related Topics

Using operators

In Corel SCRIPT, an expression is a combination of numbers, strings, variables, constants, and operators that return a result. Results can be a number, string, Boolean (TRUE or FALSE), or one of Corel SCRIPT's other data types which can be assigned to a variable. Operators are used to perform mathematical operations, concatenate strings, and logically compare expressions.

Corel SCRIPT supports the following operators:

- Arithmetic
- Bitwise
- Concatenation
- Logical
- Relational
- Unary

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

Operator precedence in Corel SCRIPT

In Corel SCRIPT, expressions with more than one operator are evaluated by a predetermined order of execution known as operator precedence. Operations with a higher precedence are followed by operations with a lower precedence. Operators with the same precedence are evaluated left to right.

Operations within parentheses are evaluated before operations outside of parentheses. Operators and expressions nested with parentheses are evaluated from the innermost to the outermost.

The following table lists the order of precedence from highest to lowest:

Order	Operator
1	parentheses ( )
2	unary +, unary - , NOT
3	exponentiation (^)
4	multiplication (*), division (/)
5	integer division (\\)
6	modulus (MOD)
7	addition (+), subtraction (-)
8	concatenation + , &
9	relational operators (=, <>, >, <, =>, >=, <=, =<)
10	AND
11	OR
12	XOR

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

Concatenation Operators

Concatenation operators link strings together.

Operator	Definition
&	Links strings or string variables together.
+	Links strings or string variables together. The + operator is also a unary and arithmetic operator.

Note

Use the STR function to return numbers as strings.

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

Examples of Concatenation Operators

& operator

```
firstString$ = "Corel"
secondString$ = "DRAW"
result$ = firstString$ & " " & secondString$
```

Assigns the value of **firstString$** followed by the value of **secondString$** to the variable **result$** with a space in between. In this example, **result$** = "Corel DRAW".

+ operator

```
firstString$ = "Corel"
secondString$ = "PHOTO-PAINT"
result$ = firstString$ & " " & secondString$
```

Assigns the value of **firstString$** followed by the value of **secondString$** to the variable **result$** with a space in between. In this example, **result$** = "Corel PHOTO-PAINT".

Using the STR function

```
a%=6
firstString$ = "Corel"
secondString$ = "DRAW"
result$ = firstString$ & " " & secondString$ & STR(a%)
```

In this example, **result$** = "Corel DRAW 6".

Note

The variables in the examples above all use type-declaration suffixes (% and $). It is not necessary to use type-declaration suffixes with these operators.

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

Unary Operators

Unary operators perform on one numeric operand.

Operator	Definition
+	Multiplies a numeric operand by +1. The + operator is also a concatenation and arithmetic operator.
-	Multiplies a numeric operand by -1. The operator is used to indicate a numeric operand is negative. The - operator is also an arithmetic operator.
NOT	Inverts the result of a logical operation. The NOT operator is also a logical and bitwise operator.

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

Examples of Unary Operators

+

```
x% = 12 / +3
```

-

```
y% = 12 / -3
```

NOT

```
a% = 5
b% = NOT(a% > 3)
```

Since **a%** is greater than 3, resulting in TRUE, **b%** is FALSE (or 0) because NOT inverts the result of the logical operation.

Note

- The variables in the examples above all use type-declaration suffixes (% and $). It is not necessary to use type-declaration suffixes with these operators.

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

Arithmetic Operators

Use arithmetic operators to perform mathematical operations on two numeric expressions. Arithmetic operators are placed between two numeric expressions.

Operator	Definition
+ (Addition)	Sums two numeric expressions. The + operator is also a concatenation and unary operator.
- (Subtraction)	Subtracts one numeric expression from another. The - operator is also a unary operator.
* (Multiplication)	Multiplies two numeric expressions.
/ (Division)	Divides two numeric expressions.
^ (Exponentiation)	Raises a numeric expression to the power of another numeric expression.
MOD (Modulus)	Returns a whole number remainder of division between two numeric expressions. Non-integer expressions are rounded before division.
\ (Integer division)	Divides two numeric operands and returns a whole number. Non-integer expressions are rounded before division to integers or longs.

Note

- All arithmetic operators have a left-associativity except the exponentiation operator (^). Left-associativity means the expression with the operator is evaluated from left to right. For example, **4 * 10 / 2** is calculated as four times ten equals 40 divided by two equals 20. For the exponentiation operator which has a right-associativity, the expression is calculated from right to left. For example, the expression 4^3^2 is calculated as 3 to the power of two (which equals 9), and then 4 to the power of 9, which equals 262,144.
- You can perform arithmetic operations on numeric expressions of different types. The return value takes on the highest precision level of the two operands. For example, if you multiply a long by an integer, the result is a long. For more information, see Explicitly declaring and assigning values to a variable.

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

Examples of Arithmetic Operators

+ (Addition)

```
i = a + b
```

Assigns the sum of the values of variables **a** and **b** to the variable **i**.

- (Subtraction)

```
j = j - 2
```

Decreases the variable **j** by 2.

*** (Multiplication)**

```
k = (c + d) * 5
```

Adds the values of variables **c** and **d** together, multiplies the sum by 5, and assigns the result to the variable **k**. Note that the parentheses force the calculation of the addition before the multiplication.

*** (Division)**

```
m = e / f
```

Assigns the result of the value of variable **e** divided by the value of **f** to the variable **m**.

^ (Exponentiation)

```
n = g ^ 2
```

Assigns the result of the value of **g** raised to the second power to the variable **n**. The expression **g ^ 2** is the same as **g * g**.

MOD (Modulus)

```
n = 14 MOD 3
n = 13.9 MOD 3.6
```

Both of the examples assign the value of 2 to **n**. In the second example, the operands are rounded to 14 and 3.

\ (Integer division)

```
n = 15 \ 4
n = 15.88 \ 4
n = 15 \ 4.123
n = 15.88 \ 4.123
```

The first and third examples assign the value of 3 to **n**. The second and fourth examples assign the value of 4 to **n**. In the second, third, and fourth examples, the operands are rounded to 16 and 4, respectively.

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

Relational Operators

The relational operators are used to compare two operands, numeric or string expressions of the same type, and return a TRUE (-1) or FALSE (0) value. Relational operators are most often used inside of Corel SCRIPT flow control structures such as DO...LOOP, IF...THEN...ELSE...ENDIF, and WHILE...WEND.

Operator	Definition
=	Equal to
<>	Not equal to
>	Greater than
<	Less than
>= (or =>)	Greater than or equal to
<= (or =<)	Less than or equal to

Note

- Because the equal sign (=) is used for both assignment and comparison, Corel SCRIPT will recognize the comparison operator only when it is within an IF, DO...LOOP, or WHILE...WEND statement.
- You can compare numeric expressions of different type (BOOL, INTEGER, SINGLE, LONG, DOUBLE). See the Corel SCRIPT data type summary for more information.
- Strings are always compared in their entirety, and case is significant. When comparing strings, the ANSI value of the characters in the strings are compared. See the Corel SCRIPT Character Map for more information.

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

Examples of Relational Operators

= (Equal to)

```
IF a% = b% THEN BEEP
```

The computer beeps if both **a%** and **b%** hold the same value. Because the equal sign (=) is used for both assignment and comparison, Corel SCRIPT will recognize the comparison operator only when it is within an IF, DO...LOOP, or WHILE...WEND statement.

<> (Not equal to)

```
IF str1$ <> str2$ THEN BEEP
```

The computer beeps if the values of the two string variables are not equal. Strings are always compared in their entirety, and case is significant. So if **str1$** is "Cat" and **str2$** is "cat", a beep will sound, because the strings are different.

> (Greater than)

```
IF a% > b% THEN BEEP
```

The computer beeps if the value of **a%** is greater than the value of **b%**.

```
IF str1$ > str2$ THEN BEEP
```

The computer beeps if the value of **str1$** is greater than the value of **str2$**. When comparing strings, the ANSI value of the characters in the strings are compared. So if **str1$** is "dog" and **str2$** is "cat", a beep will sound, because "c" comes before "d" in the ANSI character set.

< (Less than)

```
IF a% < b% THEN BEEP
```

The computer beeps if the value of **a%** is less than the value of **b%**.

```
IF str1$ < str2$ THEN BEEP
```

The computer beeps if the value of **str1$** is less than the value of **str2$**. When comparing strings, the ANSI value of the characters in the strings are compared. So if **str1$** is "cat" and **str2$** is "dog", a beep will sound, because "c" comes before "d" in the ANSI character set.

>= or => (Greater than or equal to)

```
IF a% >= b% THEN BEEP
```

The computer beeps if **a%** is greater than or equal to **b%**.

<= or =< (Less than or equal to)

```
IF a$ <= b$ THEN BEEP
```

The computer beeps if **a$** is less than or equal to **b$**.

Note

The variables in the examples above all use type-declaration suffixes (% and $). It is not necessary to use type-declaration suffixes with these operators.

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

Logical Operators

Logical operators are used in conjunction with relational operators to determine a logical relationship between expressions. Together with the relational operators, which are most often used inside of Corel SCRIPT flow control structures such as DO...LOOP, IF...THEN...ELSE...ENDIF, and WHILE...WEND, logical operators return a TRUE (-1) or FALSE (0) value.

Operator	Definition
NOT	Evaluates the expression to the right of the operator. If the expression evaluates to FALSE, TRUE is returned. The NOT operator is also a bitwise and unary operator.
AND	Logically compares the value of the expression to the left of the operator with the value of the expression to the right of the operator. If both expressions evaluate to TRUE, TRUE is returned. The AND operator is also a bitwise operator.
OR	Logically compares the value of the expression to the left of the operator with the value of the expression to the right of the operator. If either of the expressions evaluate to TRUE, TRUE is returned. The OR operator is also a bitwise operator.
XOR	Logically compares the value of the expression to the left of the operator with the value of the expression to the right of the operator. If only one of the expressions evaluates to TRUE, TRUE is returned. The XOR operator is called the exclusive operator and is also a bitwise operator.

Note

- The operators are listed in their order of precedence.

Return Values for NOT if...	Returns
Expression1 is TRUE	FALSE
Expression1 is FALSE	TRUE

Return Values for AND if...	Returns
Expression1 is TRUE and Expression2 is TRUE	TRUE
Expression1 is TRUE and Expression2 is FALSE	FALSE
Expression1 is FALSE and Expression2 is TRUE	FALSE
Expression1 is FALSE and Expression2 is FALSE	FALSE

Return Values for OR if...	Returns
Expression1 is TRUE and Expression2 is TRUE	TRUE
Expression1 is TRUE and Expression2 is FALSE	TRUE
Expression1 is FALSE and Expression2 is TRUE	TRUE
Expression1 is FALSE and Expression2 is FALSE	FALSE

Return Values for XOR if ...	Returns
Expression1 is TRUE and Expression2 is TRUE	FALSE
Expression1 is TRUE and Expression2 is FALSE	TRUE
Expression1 is FALSE and Expression2 is TRUE	TRUE
Expression1 is FALSE and Expression2 is FALSE	FALSE

Note

- The operators are listed in their order of precedence.
- Though bitwise and logical operators are listed separately, they are the same operators but function differently depending on the operand.

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

Examples of Logical Operators

NOT

```
IF NOT(a% = 0) THEN BEEP
```

If the value of **a%** is not equal to 0, a beep sounds. The paratheses indicate that NOT operates on **a% = 0** and not **a%** alone.

AND

```
IF (a% = 0) AND (b% = 0) THEN BEEP
```

If the value of **a%** is equal to 0 and the value of **b%** is equal to 0, a beep sounds. Both expressions must be true for the beep to sound.

```
IF a$ = "Corel" AND b$ = "Corel" THEN BEEP
```

If the value of **a$** is equal to "Corel" and the value of **b$** is equal to "Corel", a beep sounds. Both expressions must be true for the beep to sound.

OR

```
IF (a% = 0) OR (b% = 0) THEN BEEP
```

If either the value of **a%** is equal to 0 or the value of **b%** is equal to 0, a beep sounds. Only one of the expressions must be true for the beep to sound.

```
IF (a$ = "Corel") OR (b$ = "Corel") THEN BEEP
```

If either the value of **a$** is equal to "Corel" or the value of **b$** is equal to "Corel", a beep sounds. Only one of the expressions must be true for the beep to sound.

XOR

```
IF (a% = 0) XOR (b% = 0) THEN BEEP
```

If the value of **a%** is equal to 0 and the value of **b%** is not equal to 0, or vice versa, a beep sounds. One expression must be true and the other false for the beep to sound.

Note

- The variables in the examples above all use type-declaration suffixes (% and $). It is not necessary to use type-declaration suffixes with these operators.
- In the above examples for AND, OR, and XOR, parentheses are placed around the relational expressions but are not required because the **=** operator takes precendence over the OR operator. However, it is still good practice to put parentheses around your expressions because it makes your scripts easier to read.

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

Bitwise Operators

Use bitwise operators to compare identically positioned bits in expressions holding integers. Non-integer expressions are rounded

Operator	Definition
NOT	Unlike the three other bitwise operators, NOT is unary, meaning that it only operates on one operand (see Unary Operators for more information). NOT returns the opposite of the corresponding bits of the operand, which is sometimes called "one's-complement." The NOT operator is also a logical operator.
AND	Compares the individual bits in the expression to the left of the operator with the individual bits in the expression to the right of the operator. If the identically positioned bits are both 1, 1 is returned. If not, 0 is returned. The AND operator is also a logical operator.
OR	Compares the individual bits in the expression to the left of the operator with the individual bits in the expression to the right of the operator. If either of the identically positioned bits are 1, 1 is returned. If not, 0 is returned. The OR operator is also a logical operator.
XOR	Compares the individual bits in the expression to the left of the operator with the individual bits in the expression to the right of the operator. If the identically positioned bits are both 1 or both 0, 0 is returned. If not, 1 is returned. The XOR operator is also a logical operator.

Note

- The operators are listed in their order of precedence.
- Though bitwise and logical operators are listed separately, they are in fact the same operators but function differently depending on the operand.

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

Examples of Bitwise Operators

The following examples use only eight bits to demonstrate the bit operation, but Corel SCRIPT integers are 16-bits long. Bit positions are counted right-to-left and the first position is called the 0 position.

NOT

```
n% = NOT(a%)
```

Toggles the bit values in **a%** to set **n%**. By toggling, bits equal to 1 are set to 0 and bits equal to 0 are set to 1. For example, if **a%** = 9, **n%** = -10 as follows:

```
0 0 0 0 1 0 0 1 = 9
1 1 1 1 0 1 1 0 = -10
```

AND

```
n% = a% AND b%
```

Compares the bits of the variables **a%** and **b%** and sets the corresponding bits of **n%** to 1 if the same bit is 1 in both **a%** and **b%**. For example, if **a%** = 9 and **b%** = 12, then **n%** = 8 as follows:

```
0 0 0 0 1 0 0 1 = 9
0 0 0 0 1 1 0 0 = 12
0 0 0 0 1 0 0 0 = 8
```

Because only the 3rd bit (bits are always counted from right to left, starting with 0) is 1 in both operands, only the resulting number has the 3rd bit set to 1.

OR

```
n% = a% OR b%
```

Compares the bits of the variables **a%** and **b%**, and sets the corresponding bits of **n%** to 1 if the same bit is 1 in either **a%** or **b%**. For example, if **a%** = 9 and **b%** = 12, then **n%** = 13 as follows:

```
0 0 0 0 1 0 0 1 = 9
0 0 0 0 1 1 0 0 = 12
0 0 0 0 1 1 0 1 = 13
```

The 3rd, 2nd, and 0 bits (bits are always counted from right to left, starting with 0), are set to 1 because those bits have a value of 1 in either of the two operands.

XOR

```
n% = a% XOR b%
```

Compares the bits of the variables **a%** and **b%** and sets the corresponding bits of **n%** to 1 if the same bit is different in a% and b%. For example, if **a%** = 9 and **b%** = 12, then **n%** = 5 as follows:

```
0 0 0 0 1 0 0 1 = 9
0 0 0 0 1 1 0 0 = 12
0 0 0 0 0 1 0 1 = 5
```

Because the 3rd bit (bits are always counted from right to left, starting with 0) is 1 in both operands, the 3rd bit in the result is set to 0. The 2nd and 0 bits are different in the two operands, so those bits are set to 1 in the result.

Note

The variables in the examples above all use type-declaration suffixes (% and $). It is not necessary to use type-declaration suffixes with these operators.

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

Integer

Integers include all positive whole numbers, their negatives, and zero, e.g., -3, 0, 5.

Using strings and string functions

In Corel SCRIPT, an expression is a combination of numbers, strings, variables, constants, and operators that return a result. Results can be a number, string, Boolean (TRUE or FALSE), or one of Corel SCRIPT's other data types which can be assigned to a variable.

A string is a series of characters that is treated as a unit of information. This unit is used with certain Corel SCRIPT statements and functions to perform instructions such as saving files, accepting user input, and setting options.

Strings are a non-numeric Corel SCRIPT data type and are explicitly declared using the **$** suffix or the data type name **STRING**. They must be enclosed in double quotation marks, and can hold up to 32,765 characters including lower and uppercase letters, numbers, punctuation marks. The following example declares the string variable **stringVar$:**

```
stringVar$ = "This is a sample string."
```

Special Characters

Punctuation and certain character codes for items such as tabs and hard returns cannot be directly entered within a string's double quotation marks. The CHR function is often used to add these special characters. For example, to add double quote marks to a string variable, you use character 34:

```
s$ = CHR(34) + "This will be in double quotes." + CHR(34)
MESSAGE s$
```

You can also use the function to add a return and a line feed within a string, with character 13 and 10, respectively:

```
s$ = "String 1" + CHR(13) + CHR(10) + "String 2"
MESSAGE s$
```

The result will be the two strings on separate lines, as displayed in the message box.

See the Corel SCRIPT character map or CHR for more information about using special characters in strings.

{button ,AL(`cs_strings_fns;using_variables;corel_script_data_type_summary;;;',0,"Defaultoverview",)} Related Topics

LEFT

Returns the specified number of characters starting at the beginning of a string.

Syntax

LEFT (source$, number%)

Syntax	Definition
source$	Any string, string variable, string constant, or expression returning a string.
number%	Number of characters to be returned. Must be a non-negative number and numbers other than integers are rounded.

Note

- Subtract 1 from the result of INSTR if you do not want to include the character you are searching for.
- This function can be used to truncate user input from a dialog box.

Example

```
abc$ = LEFT("I want to dance with you", 15)
MESSAGE abc$
```

Displays "I want to dance" in a message dialog box.

Combine LEFT with INSTR to extract the portion of a string either up to or including a specified substring.

```
city$ = "San Francisco, California"
Mystr$ = LEFT(city$, INSTR(city$, ",")-1)
```

Extracts the characters in **city$** up to, but not including, the comma and places the result in the variable **Mystr$**. The variable **Mystr$** now has the value "San Francisco".

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

RIGHT

Returns the specified number of characters from the end of a string.

Syntax
RIGHT (source$, number%)

Syntax	Definition
source$	Any string, string variable, string constant, or expression returning a string.
number%	Number of characters to be returned. Must be a non-negative number and numbers other than integers are rounded.

Example

```
abc$ = RIGHT("I don't want to dance", 13)
MESSAGE abc$
```

Displays "want to dance" in a message dialog box.

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

LEN

Returns the length or number of characters in a string.

Syntax
LEN(source$)

Syntax	Definition
source$	A string variable, constant, literal, or expression.

Example

```
num% = LEN("This is a test")
```

Assigns the length of the string, 14, to the variable **num%**.

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

LTRIM

Removes any leading spaces from a string. You can use LTRIM to remove leading spaces from user dialog inputs.

Syntax
LTRIM(source$)

Syntax	Definition
source$	Any string, string variable, string constant, or expression returning a string.

Example

```
MyString$ = "   Test"
MyString$ = LTRIM(MyString$)
```

Assigns "Test" to the variable **MyString$**. All leading spaces that were previously in the variable are removed.

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

RTRIM

Removes any trailing spaces from a string. You can use RTRIM to remove trailing spaces from user dialog inputs.

Syntax
RTRIM(source$)

Syntax	Definition
source$	Any string, string variable, string constant, or expression returning a string.

Example

```
MyString$ = "Test   "
MyString$ = RTRIM(MyString$)
```

Assigns "Test" to the variable **MyString$**. All trailing spaces that were previously in the variable are removed.

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

MID

If used as a function, MID returns a specified number of characters, starting at a specified position in a string.

Syntax
MID(source$, index%, count%**)**

If used as a statement, MID replaces a portion of a string with another string, beginning at a specified character.

Syntax
MID(source$, index% ,** count%**) = modify$

Syntax	Definition
source$	Any string, string variable, string constant, or expression returning a string. Hold the string to be modified.
index%	Position of the first character to be modified.
count%	Number of characters to be modified. If not specified, the rest of **source$** is returned or overwritten.
modify$	Any string, string variable, string constant, or expression returning a string. Replaces a portion of **source$**.

Example

```
s$ = MID("I want to dance with you",11,5)
```

The function extracts five characters from the string, beginning with the eleventh character. The variable **s$** then becomes "dance".

```
str1$ = "I want to dance with you"
MID(str1$, 22, 3) = "him"
```

The statement changes three characters, starting with the 22nd character, to the new string. The **str1$** variable now contains "I want to dance with him".

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

SPACE

Returns a string consisting of a specified number of spaces (ANSI character number 32).

Syntax
SPACE (num%)

Syntax	Definition
num%	Number of spaces to be returned.

Note

- See the Corel SCRIPT Character Map for more ANSI details and Windows characters.

Example

```
Mystr$ = SPACE(4) + "Test" + SPACE(4)
```

Makes the string variable **Mystr$** equal to " Test ". (The string **Mystr$** now consists of 4 spaces, the word TEST and another 4 spaces).

```
x1 = "Corel"
x2 = "SCRIPT"
x3 = x1 + SPACE(1) + x2
```

Makes the string variable **x3$** equal to "Corel SCRIPT".

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

LCASE

Converts a string to lowercase.

Syntax
LCASE(source$)

Syntax	Definition
source$	Any string, string variable, string constant, or expression returning a string.

Example

```
x$="HI"
firststring$ = LCASE(x)
secondstring$ = LCASE("THerE")
MESSAGE firststring + " " + secondstring
```

Displays the converted strings "hi there" in a message dialog box.

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

UCASE

Converts a string to uppercase.

Syntax
UCASE(source$)

Syntax	Definition
source$	Any string, string variable, string constant, or expression returning a string.

Example

```
x$="hI"
firststring$ = UCASE (x)
secondstring$ = UCASE("THerE")
MESSAGE firststring + " " + secondstring
```

Displays the converted strings "HI THERE" in a message dialog box.

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

Transformation commands

VAL

Converts a string to a number. The number's variable type is double. This function is the opposite of the STR statement.

Syntax
VAL (chars$)

Syntax	Definition
chars$	The string to be converted. If the string does not begin with a number, VAL returns 0.

Note

- You can only use the period as a decimal separator with the VAL function. If you're not using a period (**.**) as a decimal separator, the CDBL or CSNG function can be used to convert a string to a number. Your Windows decimal settings are set in the Control Panel's Regional Settings.
- Because text box controls in dialog boxes can only return strings (even if numbers are inputted), you can use the VAL function to convert strings entered in dialog boxes to numbers.

Example

```
g = VAL("72")
h = VAL("72.700113")
```

Both the above statements assign 72 to the variables **g** and **h**.

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

ASC

Returns the numerical ANSI character value of the first character specified in a string literal, string constant, or string variable. ASC is the opposite of the CHR function, which returns a character when the ANSI value is specified.

Syntax
ASC(source$)

Syntax	Definition
source$	The string to be examined.

Note

- See the Corel SCRIPT Character Map for more ANSI details and Windows characters.

Example

```
i% = ASC("string")
```

This expression will assign the value 115, which is the ANSI value of the letter "s."

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

CHR

Returns the ANSI character that occupies the specified position in the ANSI character set. CHR is the opposite of ASC, which returns the ANSI code value when the character is entered. See the Corel SCRIPT Character Map for more ANSI details and Windows characters.

Syntax
CHR (value%)

Syntax	Definition
value%	ANSI code value to be examined.

Example

```
s$ = CHR(65)
```

Assigns the letter "A" to the variable **s$**. (Character 65 of the ANSI character set is A.)

Special Characters

The CHR function is often used to add special characters to string variables that cannot be entered directly within double quotes. For example, to add double quote marks to a string variable, you use character 34:

```
s$ = CHR(34) + "This will be in double quotes." + CHR(34)
MESSAGE s$
```

You can also use the function to add a return and a line feed within a string, with character 13 and 10, respectively:

```
s$ = "String 1" + CHR(13) + CHR(10) + "String 2"
MESSAGE s$
```

The result will be the two strings on separate lines, as displayed in the message box.

The following table notes some of the special characters you can use with the CHR function.

Character Number	Special Character Returned
8	Backspace
9	Tab
10	Linefeed
13	Return
32	Space
34	Quotation mark

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

STR

Returns a string representation of a numeric data type. The STR function is useful when you want to manipulate a number as a string.

Syntax
STR(num)

Syntax	Definition
num	Any number, numeric variable, constant, or expression.

Note

- You can only use the period as a decimal separator with the STR function. If you're not using a period (**.**) as a decimal separator, the CSTR function can be used to convert a number to a string. Your Windows decimal settings are set in the Control Panel's Regional Settings.
  If you're converting dates, they must be in the standard U.S. format (M/d/yy h:mm:ss TT).
- If a positive number is converted, the STR function inserts a leading space before the first character. If a negative number is converted, the STR function inserts a negative sign before the first character.

Example

```
aInteger$ = STR(72)
aNonInteger$ = STR(.140166)
```

The first example assigns "72" to the variable **aInteger$**. The second example assigns "0.140166" to **aNonInteger$**.

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

INSTR

Returns the starting position of the first occurrence of a string within another string. If the specified string is not found, the function returns 0.

Syntax

INSTR (string1$, string2$, start%**)**

Syntax	Definition
string1$	String within which the search is made.
string2$	String searched for.
start%	Specifies the position where the search begins within **string1$**. If unspecified, the search starts at the beginning of **string1$** (same as **start%** = 1). Must be a non-negative number and numbers other than integers are rounded.

Example

```
pos% = INSTR("Los Angeles", "Ang")
```

Sets **pos%** to the value 5 because "Ang" occurs at the fifth character in the string "Los Angeles".

```
pos% = INSTR("Los Angeles: City of Angels", "Ang", 8)
```

Sets **pos%** to the value 22.

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

Using user-interface statements and commands

You can use user interface statements to define and display several kinds of interactive dialog boxes without having to design and create a user-defined dialog box.

Many times you need to get information from the user before your script performs a desired action. For simple information, you can use the Corel SCRIPT function INPUTBOX to get a string from the user returned to a running script. If you want to provide the user with options and more complex information, such as a list of choices, you can include a user-defined dialog box in your script.

Note

- The dialog boxes created using the statements INPUTBOX, MESSAGEBOX, GETFILEBOX, and any user-defined dialog boxes that include a Cancel button involve a system variable called CANCEL. When the script starts, this variable is set to zero. If the Cancel button is chosen, the variable is set to TRUE. The value of the CANCEL variable remains unchanged between statement calls, unless the script changes it. A typical use of this variable is:

```
IF CANCEL THEN STOP
```

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

BEEP

Sounds a tone.

Syntax
BEEP

Note

- The sound your computer makes depends on your computer's hardware (i.e., sound cards, PC speaker, ect) and the Default sound in your Windows sound settings (see your Windows Control Panel for more details).

Example

```
IF (abc<=15.3) then BEEP ELSE MESSAGE "It's greater than 15.3"
```

If the variable **abc** is less than or equal to 15.3, the computer sounds a tone.

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

GETFILEBOX

Displays a standard Windows File Open or File Save dialog box. Both dialog boxes allow users to choose a file from the file system. GETFILEBOX returns the selected filename and its full path, or an empty string if the user chooses Cancel. The GETFILEBOX statement by itself does not open or save a file; it only returns a string.

Syntax

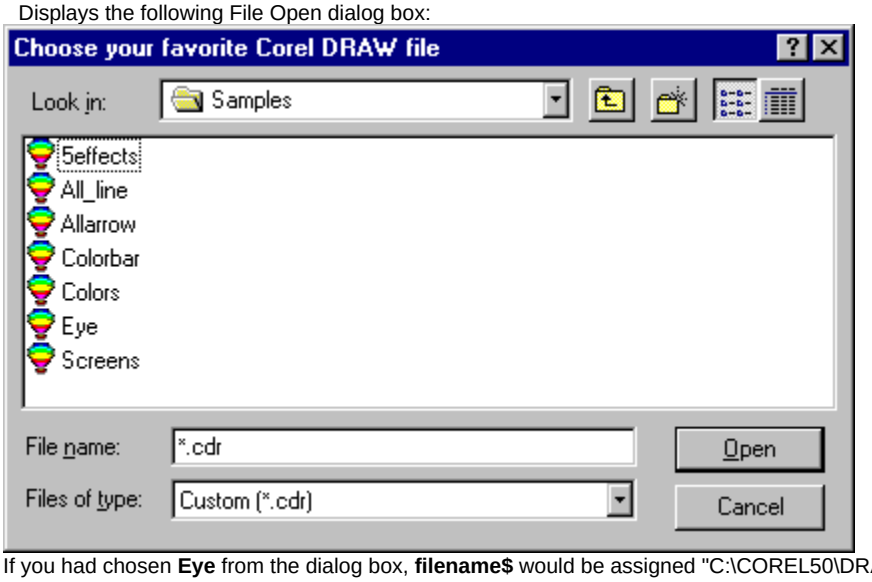
GETFILEBOX (filespec$, title$, type%**)**

Syntax	Definition
filespec$	Specifies the files to display in the box. This can include a default folder and wild-card characters. You can separate multiple extensions with a semicolon (;).
title$	The title to display in the dialog box.
type%	Specifies the type of dialog box to display: 0 = File Open dialog box (default if omitted) 1 = File Save dialog box

Example

```
CURRFOLDER = "c:\COREL60\DRAW\samples" 'set the current folder
filename$ = GETFILEBOX("*.cdr", "Choose your favorite Corel DRAW file", 0)
```

Displays the following File Open dialog box:



If you had chosen **Eye** from the dialog box, **filename$** would be assigned "C:\COREL50\DRAW\SAMPLES\Eye.cdr"

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

INPUTBOX

Displays a simple dialog box where you can enter a string. The dialog box has OK and Cancel buttons. If the Cancel button is chosen, an empty string is returned.

Syntax
INPUTBOX(prompt$)

Syntax	Definition
prompt$	String, string variable, constant, or expression that appears in the dialog box above the edit box.

Example

```
MyString$ = INPUTBOX("Please type in a string")
```

User input is returned to the variable **MyString$.**

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

MESSAGE

Displays a box containing a specified message and an OK button.

Syntax
Message anyVariable

Syntax	Definition
anyVariable	Any constant, string, string variable, number, numeric variable or expression to display in the message box. Numbers, numeric variables, dates, and expressions are displayed as their string representations.

Example

```
x$="Hello." + CHR(13)   'CHR(13) is a return character
MESSAGE x$ + "What a nice day."
```

Note

- See the Corel SCRIPT character map for a list of character codes.

{button ,AL(`cs_ui_statements;textbox;text;chr;;',0,"Defaultoverview",)} Related Topics

MESSAGEBOX

Displays a message box with a specified message and user-specified buttons and icons.

Syntax
MESSAGEBOX(prompt$, title$, option%)

Syntax	Definition
prompt$	The string to display in the box.
title$	The string to display in the message box caption.
option%	A number representing the type of buttons to include in the box and an icon (if any) to appear beside the message. The **option%** value is set using the OR (or +) operator for multiple buttons (see example):

Button Type

0	OK only
1	OK/Cancel
2	Abort/Retry/Ignore
3	Yes/No/Cancel
4	Yes/No
5	Retry/Cancel

Icon Type

0	No icon
16	Stop
32	Question
48	Exclamation
64	Information (lowercase i)

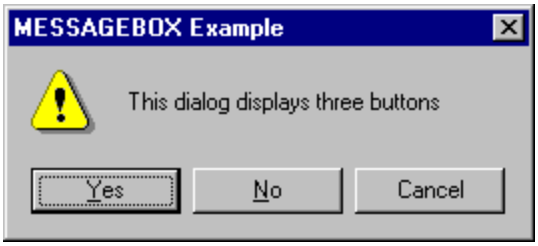
Returns

The number associated with the button pressed by the user:

1	OK
2	Cancel
3	Abort
4	Retry
5	Ignore
6	Yes
7	No

Example

```
retval% = MESSAGEBOX("This dialog displays three buttons", "MESSAGEBOX Example", 3 OR 48)
```



{button ,AL(`cs_ui_statements;textbox;text;;;',0,"Defaultoverview",)} Related Topics

CONST

Declares symbolic constants for use in place of Corel SCRIPT data values. Constants can be useful in mathematical functions, and for values that may be used several times in a script.

Syntax

GLOBAL **CONST constant**{%|&|!|#|@|$} **= expression** , constant{%|&|!|#|@|$} = expression, ...

Syntax	Definition
GLOBAL	An optional parameter used to declare global constants. Global constants are available to all procedures in a script (main section, subroutines, or functions). If not used, the constant is only available to the procedure in which it was declared.
constant{%|&|!|#|@|$}	Specifies the name of the constant and follows the Corel SCRIPT naming convention. Optionally, a type-declaration suffix can follow the name.
expression	The expression is a combination of numbers, strings, variables, constants, and operators that return a result.

Note

- Constants can be used to declare the size of an array.
- Changing the value of a constant normally only requires editing one script statement.

Example

```
REM creates a global constant for the base of the natural logarithm
GLOBAL CONST NATURAL_LOG# = 2.71828182845

REM creates a local constant for pi
CONST PI = 3.14
```

{button ,AL(`script_procedures;global;dim;using_variables;Using_constants;corel_script_data_type_summary;;;',0,"Defaultoverview",)} Related Topics

DECLARE

Before a script can call a user-defined function or subroutine, it must be declared with the DECLARE statement. This specifies the name of the user-defined procedure and the number and type of arguments passed to it. DECLARE is required for all user-defined functions and subroutines. The DECLARE statement must be placed at the beginning of a script; that is, it can only be preceded by remark statements or other DECLARE statements.

Syntax

DECLARE FUNCTION name{%|&|!|#|@|$} (**PASS parameter{%|&|!|#|@|$}, PASS parameter{%|&|!|#|@|$}, ...)**
DECLARE FUNCTION name (PASS parameter AS type, PASS parameter AS type, ...**) AS returnType**
DECLARE SUB name (PASS parameter{%|&|!|#|@|$}, PASS parameter{%|&|!|#|@|$}, ...**)**
DECLARE SUB name (PASS parameter AS type, PASS parameter AS type, ...**)**

Syntax	Definition
FUNCTION	Used to declare a function.
SUB	Used to declare a subroutine.
name{%|&|!|#|@|$}	Name of the function and follows the Corel SCRIPT naming convention. A type-declaration suffix must follow the name to set the function's data type. The name must correspond to the name of the procedure in the same script file.
name	Name of the function or subroutine and follows the Corel SCRIPT naming convention. The name must correspond to the name of the procedure in the same script file.
AS type	Declares the function or parameter type with a type declaration name.
parameter{%|&|!|#|@|$}	Name of the function parameter(s) and follows the Corel SCRIPT naming convention. A type-declaration suffix must follow the name to set the data type returned to the function.
parameter	Name of the function parameter(s) and follows the Corel SCRIPT naming convention.
PASS	PASS determines how the variable it precedes in the script is passed to the procedure. PASS can be set to BYVAL or BYREF. When set to BYVAL, the value of the variable it precedes is passed by value. That is, the procedure accesses a copy of the variable and its value can't be changed by the procedure to which it was passed. When set to BYREF, the value of the variable it precedes is passed by reference and its value can be changed by the procedure. PASS is optional, and if omitted, Corel SCRIPT uses BYREF.
AS returnType	Declares the data type of the value returned by the function with a type declaration name.

Note

- Subroutines and user-defined functions must be declared in the main section of a script file.
- Use the DECLARE...LIB statement to call functions and subroutines in Windows Dynamic Link Libraries (DLL files).

{button ,AL(`using_functions_subroutines;call;function_end_function;sub_end_sub;;',0,"Defaultoverview",)} Related Topics

Example for DECLARE and FUNCTION statements

```
REM main section of script file
DECLARE FUNCTION cube_function%(a%) 'function declaration
DECLARE FUNCTION rank_function(a as INTEGER, b as STRING) as STRING
'
'
MESSAGE CSTR(cube_function(3))
MESSAGE rank_function(1, "Corel Script")
'
'
REM function section of script file
FUNCTION cube_function%(a%) 'function definition
    cube_function = a^3
END FUNCTION
'
FUNCTION rank_function(a as INTEGER, b as STRING) as STRING
    rank_function = b + " is #"+ CSTR(a)
END FUNCTION
```

In the above example, two functions are declared and defined. The first function cubes an integer and the second function creates a string. The result of each function is displayed in a message dialog box.

{button ,AL(`example_vars;function_end_function;sub_end_sub;declare;;',0,"Defaultoverview",)} **Related Topics**

Example for DECLARE, SUB, and CALL statements

```
REM main section of script file
DECLARE SUB cube_sub(a%) 'sub declaration
DECLARE SUB rank_sub(a as INTEGER, b as STRING)
'
'
CALL cube_sub(3)
rank_sub(1, "Corel Script") 'calling without call statement
'
'
REM subroutine section of script file
SUB cube_sub(a%) 'sub definition
    MESSAGE CSTR(a^3)
END SUB
'
SUB rank_sub(a as INTEGER, b as STRING)
    MESSAGE b + " is #"+ CSTR(a)
END SUB
```

In the above example, two subroutines are declared and defined. The first subroutine, **cube_sub**, displays the result of cubing the subroutine's parameter in a message dialog box. The second subroutine, **rank_sub**, displays the result of concatenating strings in a message dialog box.

The **cube_sub** is called with the CALL statement while **rank_sub** is not.

{button ,AL(`example_vars;function_end_function;sub_end_sub;declare;call;',0,"Defaultoverview",)} Related Topics

DECLARE...LIB

Corel SCRIPT scripts can be used to call functions and subroutines in Windows Dynamic Link Libraries (DLL files).

To call an external procedure, it must first be declared with the DECLARE...LIB statement. The statement specifies the name of a procedure and the DLL that contains the procedure, and it defines the number and type of arguments passed to it. The DECLARE...LIB statement must be placed at the beginning of a script; that is, it can only be preceded by remark statements or other DECLARE statements.

Consult the DLL's technical reference before you call any of its functions. For example, to use the Windows DLLs, you need the *Windows Software Development Kit*.

Warning

You should save or back up essential files and programs before using functions and subroutines in DLL files. Passing invalid arguments to a function can result in a Windows General Protection Fault or unstable system behavior.

Syntax for declaring functions

DECLARE FUNCTION procName LIB "file" (PASS argument AS type, PASS argument AS type, ...**)** AS returnType ALIAS "aliasName"

Syntax for declaring subroutines

DECLARE SUB procName LIB "file" (PASS argument AS type, PASS argument AS type, ...**)** ALIAS "aliasName"

Syntax	Definition
FUNCTION	Use for procedures that return as value that can be assigned to an expression.
SUB	Use for procedures that do not return a value.
procName	String specifying the name of the called procedure; must correspond to the procedure name in the DLL module where it resides; name matching is case-sensitive. You have the option of using another name for the procedure if you specify the procedure's name using the ALIAS syntax part of the statement.
"file"	String specifying the DLL file to access. You should specify the DLL extension and path; if not specified, Corel SCRIPT searches for the file in the Windows folder with a DLL extension. Quotation marks must be used.
argument	Variables that pass values to the procedure when it is called. You can use more than one argument but each argument's data type must be declared. Some procedures don't pass arguments but the brackets ( ) must still be used.
AS type	Declares an argument's type with a type declaration name. If the procedure uses an argument, a type declaration name must be used.
AS returnType	Declares the data type of the value returned by the function with a type declaration name.
ALIAS "aliasName"	String specifying the name of the called procedure; must correspond to the procedure name in the DLL module where it resides; name matching is case-sensitive. This option must be used if the DLL **procName** is not in uppercase characters. You should also use an alias if the actual name of the procedure is a reserved Corel SCRIPT keyword. Quotation marks must be used.
PASS	PASS determines how the variable it precedes in the script is passed to the procedure. PASS can be set to BYVAL or BYREF. When set to BYVAL, the value of the variable it precedes is passed by value. That is, the procedure accesses a copy of the variable and its value can't be changed by the procedure to which it was passed. When set to BYREF, the value of the variable it precedes is passed by reference and its value can be changed by the procedure. PASS is optional, and if omitted, Corel SCRIPT uses BYREF.

Note

- You can declare an arguments data type using a Corel SCRIPT type declaration suffix. For example: `X$` instead of `X AS STRING`
- If a DLL function or subroutine uses an unsupported variable type, you might not be able to use it in a script. See Corel SCRIPT data type summary for a list of supported data types. Declare unsupported data types as a Corel SCRIPT data types if the unsupported data type uses the same number of bytes as a Corel SCRIPT data type.
- The DECLARE statement can be placed anywhere in a script before the first subroutine or function call.

{button ,AL(`Using_Dynamic_Link_Libraries;getapphandle;getwinhandle;declare;function_end_function;sub_end_sub;call;Corel_SCRIPT_advanced_programming_features;',0,"Defaultoverview",)} Related Topics

Examples of DECLARE...LIB statement

```
DECLARE FUNCTION GetActiveWindow LIB "C:\WINDOWS\user.exe" ( ) AS INTEGER
DECLARE SUB CloseWindow LIB "C:\WINDOWS\user.exe" (win AS INTEGER)
```

The first statement defines a procedure named **GetActiveWindow** (a Windows function that does not require a type-declaration character). The executable code for this procedure is stored in "user.exe" which is actually a Dynamic Link Library without a DLL extension. The ( ) indicate an empty parameter list, and the clause AS INTEGER describes the type of value the function returns.

The second statement is similar, except that a SUB procedure does not return a value. The parameter's data type is declared with an AS clause, as shown by **(win AS INTEGER)**.

The following are some examples of the DECLARE...LIB statement using Windows DLL files:

```
DECLARE FUNCTION FindWindow LIB "user32" (BYVAL classname AS LONG,BYVAL title AS STRING) AS
INTEGER ALIAS "FindWindowA"
```

Looks for a given window title and class. Returns the handle of that window

```
DECLARE FUNCTION SetFocus LIB "user32" (BYVAL hwnd AS INTEGER) AS INTEGER ALIAS "SetFocus"
```

Gives focus to a specified window

```
DECLARE FUNCTION WinExec LIB "kernel32" (BYVAL path AS STRING,BYVAL shw AS INTEGER) AS INTEGER
ALIAS "WinExec"
```

Executes the specified exe.

```
DECLARE FUNCTION GetActiveWindow LIB "user32" () AS INTEGER ALIAS "GetActiveWindow"
```

Returns the handle of the active window

```
DECLARE SUB CloseWindow LIB "user32" (BYVAL win AS INTEGER) ALIAS "CloseWindow"
```

Closes the specified window.

{button ,AL(`declare;function_end_function;sub_end_sub;call;Corel_SCRIPT_advanced_programming_features;',0,"Defaultoverview",)} Related Topics

DIM

Use DIM to explicitly declare and assign values to local variables or to specify the number and type of elements in an array.

Syntax for variables

**DIM variable{%|&|!|#|@|$}**, variable{%|&|!|#|@|$} ,...
DIM variable AS type, variable AS type,...

Syntax for arrays

DIM array_name{%|&|!|#|@|$} (upperbound)
DIM array_name(upperbound) AS type
DIM array_name{%|&|!|#|@|$} (lowerbound TO upperbound)
DIM array_name(lowerbound TO upperbound) AS type

Syntax	Definition
variable{%|&|!|#|@|$}	Specifies the name of the variable and follows the Corel SCRIPT naming convention. A type-declaration suffix must follow the name.
variable	Name of the variable and follows the Corel SCRIPT naming convention.
array_name{%|&|!|#|@|$}	Specifies the name of the array and follows the Corel SCRIPT naming convention. A type-declaration suffix must follow the name in the case of an array.
array_name	Specifies the name of the array and follows the Corel SCRIPT naming convention.
upperbound	The upper bound of the array expressed as an integer. If you do not use a TO clause to specify the number of array elements, the default (1 TO **upperbound**) is used.
lowerbound	The lower bound of the array expressed as an integer. If you do not use a TO clause to specify the number of array elements, the default (1 TO **upperbound**) is used.
type	Declares the variable's or array's type with a type declaration name.

Note

- Variables declared in the main section of a script are only available in the main section. Global variables are available to all procedures in a script (main section, subroutines, or functions). See GLOBAL for more information.
- Declaring a variable in a subroutine or a function makes it available only in the procedure it was declared.
- It's a generally accepted programming convention to put declaration statements at the beginning of a procedure (main section, subroutines, or functions).
- The DIM statement can be placed anywhere in a script before the variable(s) it declares is called.
- Arrays can only hold one data type. The number of elements arrays can hold is limited to your computer memory.
- You can't change the number of elements in an array once it has been declared.
- See Multi-dimensional arrays to create arrays of more than one dimension.

{button ,AL(`dim;global;using_arrays;using_variables;lbound;ubound;multi_dimensional_arrays;',0,"Defaultoverview",) } Related Topics

GLOBAL

Use GLOBAL to explicitly declare and assign values to variables or to specify the number and type of elements in an array. Global variables and arrays are available to all procedures in a script (main section, subroutines, or functions). See the DIM statement about declaring local variables.

Syntax for variables

**GLOBAL variable{%|&|!|#|@|$}**, variable{%|&|!|#|@|$} ,...
GLOBAL variable AS type, variable AS type,...

Syntax for arrays

GLOBAL array_name{%|&|!|#|@|$} (upperbound)
GLOBAL array_name(upperbound) AS type
GLOBAL array_name{%|&|!|#|@|$} (lowerbound TO upperbound)
GLOBAL array_name(lowerbound TO upperbound) AS type

Syntax	Definition
variable{%|&|!|#|@|$}	Specifies the name of the variable and follows the Corel SCRIPT naming convention. A type-declaration suffix must follow the name.
variable	Name of the variable and follows the Corel SCRIPT naming convention.
array_name{%|&|!|#|@|$}	Specifies the name of the array and follows the Corel SCRIPT naming convention. A type-declaration suffix must follow the name in the case of an array.
array_name	Specifies the name of the array and follows the Corel SCRIPT naming convention.
upperbound	The upper bound of the array expressed as an integer. If you do not use a TO clause to specify the number of array elements, the default (1 TO **upperbound**) is used.
lowerbound	The lower bound of the array expressed as an integer.
type	Declares the variable's or array's type with a type declaration name.

Note

- Global variables cannot be declared in a subroutine or a function. Additionally, globals cannot be declared in a flow construct such as FOR...NEXT or DO...LOOP.
- It's a generally accepted programming convention to put declaration statements at the beginning of the script.
- Arrays can only hold one data type. The number of elements arrays can hold is limited to your computer memory.
- You can also declare global constants. See CONST for more information.
- See Multi-dimensional arrays to create arrays of more than one dimension.

{button ,AL(`const;dim;global;using_arrays;using_variables;lbound;ubound;multi_dimensional_arrays;',0,"Defaultoverview",)} Related Topics

Examples for DIM statement

Variables

```
DIM my_color$
DIM my_color AS STRING
```

The above examples show different methods of declaring variables. The above DIM statements all declare strings.

```
DIM a AS INTEGER, b AS BOOLEAN, c AS SINGLE
```

You can also mix the type of variables you declare with a DIM statement.

Arrays

```
DIM color$(5)
color$(1) = "black"
color$(2) = "red"
color$(3) = "white"
color$(4) = "blue"
color$(5) = "green"
```

Creates a string array named **color$** that consists of 5 elements.

```
DIM salespeople(-2 TO +3) AS INTEGER
salespeople(-2) = 1
salespeople(-1) = 3
salespeople(0) = 5
salespeople(1) = 7
salespeople(2) = 9
salespeople(3) = 11
```

Creates an integer array named salespeople that consists of 6 elements.

Note

- See Multi-dimensional arrays to create arrays of more than one dimension.

{button ,AL(`example_multi_array;dim;global;using_variables;using_arrays;',0,"Defaultoverview",)} Related Topics

Examples for GLOBAL statement

Variables

```
GLOBAL my_color$
GLOBAL my_color AS STRING
```

The above examples show different methods of declaring global variables. The above GLOBAL statements all declare strings.

```
GLOBAL a AS INTEGER, b AS BOOLEAN, c AS SINGLE
```

You can also mix the type of variables you declare with a GLOBAL statement.

Arrays

```
GLOBAL color$(5)
color$(1) = "black"
color$(2) = "red"
color$(3) = "white"
color$(4) = "blue"
color$(5) = "green"
```

Creates a global string array named **color$** that consists of 5 elements.

```
GLOBAL salespeople(-2 TO +3) AS INTEGER
salespeople(-2) = 1
salespeople(-1) = 3
salespeople(0) = 5
salespeople(1) = 7
salespeople(2) = 9
salespeople(3) = 11
```

Creates an global integer array named salespeople that consists of 6 elements.

Note

- See Multi-dimensional arrays to create arrays of more than one dimension.

{button ,AL(`example_multi_array;dim;global;using_variables;using_arrays;',0,"Defaultoverview",)} Related Topics

FUNCTION and END FUNCTION

FUNCTION is used in the first line of a user-defined function and END FUNCTION is the last line. You have the option of using type-declaration characters and names in the FUNCTION statement. However, if you do use type-declaration characters and names, the syntax must replicate the syntax used with the function's DECLARE statement. Parentheses are required.

Syntax

FUNCTION name{%|&|!|#|@|$} **(**parameter{%|&|!|#|@|$}, parameter{%|&|!|#|@|$}, ...**)**
    [statements]
END FUNCTION

    or

FUNCTION name (parameter AS type, parameter AS type, ...**)** AS type
    [statements]
END FUNCTION

Syntax	Definition
name{%|&|!|#|@|$}	Name assigned to the function and follows the Corel SCRIPT naming convention. The name must correspond to the name used in the DECLARE statement. A type-declaration suffix must follow the name and be of the same type used in the DECLARE statement, if applicable.
name	Name assigned to the function and follows the Corel SCRIPT naming convention. The name must correspond to the name used in the DECLARE statement.
[statements]	Script instructions that are executed when the function is called.
parameter{%|&|!|#|@|$}	Variable(s) to store the value(s) passed to the function. The variables follow the Corel SCRIPT naming convention. A type-declaration suffix can be used to set the data type passed to the function.
parameter	Variable(s) to store the value(s) passed to the function. The variables follow the Corel SCRIPT naming convention.
AS type	Declares the function or parameter type with a type declaration name .

Note

- Subroutines and user-defined functions are both Corel SCRIPT procedures that execute instructions such as creating and modifying variables. Additionally, functions can also be used to return values.
- The parameters must be listed in the FUNCTION statement in the same order as in the function's DECLARE statement.
- It's a generally accepted programming convention to indent function statements.
- Use the DECLARE...LIB statement to declare functions and subroutines in Windows Dynamic Link Libraries (DLL files).

{button ,AL(`using_functions_subroutines;declare;call;;;',0,"Defaultoverview",)} Related Topics

LBOUND

Returns the lower bound for the indicated dimension of an array. If the dimension is omitted, the lower bound of the first dimension is returned.

Syntax

LBOUND(array{%|&|!|#|@|$},dimension**)**

Syntax	Definition
array{%|&|!|#|@|$}	The name of the array being checked.
dimension	An integer variable or numeric constant ranging from 1 to the number of dimensions in the array. Indicates which dimension's lower bound is returned; if omitted, the limit of the first dimension is returned.

Example

```
DIM a%(-5 TO 7, 10)
x% = LBOUND(a%,1)
y% = LBOUND(a%,2)
```

Sets x% and y% to -5 and 1, respectively.

{button ,AL(`LBOUND;UBOUND;DIM;USING_ARRAYS;multi_dimensional_arrays;',0,"Defaultoverview",)} Related Topics

LET

Assigns the value of an expression to a variable. The **LET** keyword is optional.

Syntax

LET **variable**{%|&|!|#|@|$} **= expression**
LET **variable = expression** AS type
variable{%|&|!|#|@|$} **= expression**
variable = expression AS type

Syntax	Definition
variable{%|&|!|#|@|$}	Name of variable and is assigned **expression**'s value. The variable name follows the Corel SCRIPT naming convention.
variable	Name of variable and is assigned **expression**'s value. The variable name follows the Corel SCRIPT naming convention.
expression	The expression is a combination of numbers, strings, variables, constants, and operators that return a result.
type	Declares the variable's type with a type declaration name.

Note

- There isn't an advantage in using the LET statement to assign an expression to a variable, but in some cases it can make your script easier to read and modify.

Example

```
LET stringVar$ = "This is a string."
```
Assigns the string "This is a string." to the variable stringVar$.

```
stringVar$ = "This is a string."
```
Assigns the string "This is a string." to the variable stringVar$. The LET keyword is omitted.

```
result% = (a% + b%) / c%
```
Assigns the result of the sum of the values of variables a% and b%, divided by the value of c%, to the variable result%. The LET keyword is omitted.

{button ,AL(`variable_availability;using_variables;Dim;;;',0,"Defaultoverview",)} Related Topics

STATIC

Used to declare variables in a subroutine or a user-defined function. Static variables retain their values after a subroutine or a function terminates, and the retained value can be used by the script the next time the subroutine or function is called.

Syntax

STATIC variable{%|&|!|#|@|$}
STATIC variable AS type

Syntax	Definition
variable{%|&|!|#|@|$}	A variable name following the Corel SCRIPT naming convention.
variable	A variable name following the Corel SCRIPT naming convention.
type	Declares the constant's type with a type declaration name.

Note

- It's a generally accepted programming convention to put static declaration statements at the beginning of subroutines or functions with DIM declarations.
- Static are always initialized to 0.

{button ,AL(`variable_availability;using_variables;Dim;;;',0,"Defaultoverview",)} Related Topics

Example for STATIC statement

```
REM main section of script file
DECLARE FUNCTION staticFunc% (a%)
FOR i% = 1 to 5
    j% = staticFunc%(i%)
NEXT I%
'
REM (Static Function Example)
FUNCTION staticFunc%(a%)
STATIC staticVar%
' Because staticVar% is STATIC, it retains its previous value
' each time the function is called
staticVar% = staticVar% + a%
' The function returns the current value of staticVar%
staticFunc% = staticVar%
END SUB
```

The variable **staticVar%** in the function is created as a STATIC variable, so that its value remains unchanged each time the function is called. In the main program, a FOR loop calls the function five times. The result of each function call follows:

1 The first time the script runs, **staticVar%** has a value of 0 because it is created for the first time. The passed parameter, **i%**, has a value of 1, and the variable also has a value of 1.

2 In the second call, **staticVar%** has a value of 1 and the passed parameter has a value of 2. So the calculation causes **staticVar%** to be 3.

3 In the third call, **staticVar%** is equal to 3 and **i%** is equal to 3, so **staticVar%** has a new value of 6.

4 In the fourth call, **staticVar%** is 6 and **i%** is 4, giving **staticVar%** a new value of 10.

5 In the last call, **staticVar%** is 10 and **i%** is 5, giving **staticVar%** a value of 15.

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

SUB...END SUB

SUB is used in the first line of a subroutine definition and END SUB is the last line. Use parentheses if arguments are present.

Syntax

SUB name (parameter{%|&|!|#|@|$}, parameter{%|&|!|#|@|$}, ...)
    [statements]
END SUB

    or

SUB name (parameter AS type, parameter AS type, ...**)**
    [statements]
END SUB

Syntax	Definition
name	Name assigned to the subroutine and follows the Corel SCRIPT naming convention. The name must correspond to the name used in the DECLARE statement.
parameter{%|&|!|#|@|$}	Variable(s) to store the value(s) passed to the subroutine procedure. The variables follow the Corel SCRIPT naming convention. A type-declaration suffix must follow the name to set the data type passed to the subroutine.
parameter	Variable(s) to store the value(s) passed to the subroutine procedure. The variables follow the Corel SCRIPT naming convention.
[statements]	Script instructions that are executed when the subroutine is called.
type	Declares the parameter type with a type declaration name and must be used if a parameter is not using a type-declaration suffix.

Note

- Subroutines and user-defined functions are both Corel SCRIPT procedures that execute instructions such as creating and modifying variables. Additionally, functions can also be used to return values.
- The parameters must be listed in the SUB statement in the same order as in the subroutine's DECLARE statement.
- It's a generally accepted programming convention to indent subroutine statements.
- Use the DECLARE...LIB statement to declare functions and subroutines in Windows Dynamic Link Libraries (DLL files).

{button ,AL(`using_functions_subroutines;declare;call;;;',0,"Defaultoverview",)} Related Topics

UBOUND

Returns the upper bound for the indicated dimension of an array. If dimension is omitted, the upper bound of the first dimension is returned.

Syntax

UBOUND(array{%|&|!|#|@|$},dimension**)**

Syntax	Definition
array{%|&|!|#|@|$}	The name of the array being checked.
dimension	An integer variable or numeric constant ranging from 1 to the number of dimensions in the array; indicates which dimension's lower bound is returned. If omitted, the limit of the first dimension is returned.

Example

```
DIM a%(-5 TO 7, 10)
x% = UBOUND(a%,1)
y% = UBOUND(a%,2)
```

Sets x% and y% to 7 and 10, respectively.

{button ,AL(`LBOUND;UBOUND;DIM;USING_ARRAYS;multi_dimensional_arrays;',0,"Defaultoverview",)} Related Topics

Using Variables

In Corel SCRIPT, an expression is a combination of numbers, strings, variables, constants, and operators that return a result. Results can be a number, string, Boolean (TRUE or FALSE), or one of Corel SCRIPTs other data types which can be assigned to a variable.

A variable is a value place holder whose name points to an address in the computer's memory where that value is stored. Variable addresses can only hold one value at a time, but the value can change when a script is run by sending the variable a script instruction. Once a script terminates, a variable and the value it's holding is lost.

Variable names

Variable names can be made up of any combination ofletters in the alphabet, both lowercase and uppercase, the numbers 0 through 9, and the underscore character (_). Other rules for variable names follow:

- Variable names are not case sensitive. For example, the variable **ABC** is the same as **abc**, **Abc**, **aBC**, and so on**.**
- The initial character must be a letter or the underscore character and can be followed by any combination of letters, numbers, and the underscore character. Variable names cannot include spaces. The following are all valid variable names:

  x, X123, Corel_6, TheVariable, a1B2, This_is_a_variable, ThisIsAVariable
- The maximum length of a variable's name is 256 characters.
- A variable cannot have the same name as a Corel SCRIPT statement, function, or operator. See Reserved Words for a complete listing of words that can't be used as a variable name.
- All variable names must be unique in a script procedure.

Variable Types

Corel SCRIPT supports eight different variable types: seven numeric types (including a Boolean type and date type) and a string type. A variable's type sets the type of data a variable can hold. For more information about data types, see Corel SCRIPT data type summary.

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

Variable availability

Corel SCRIPT scripts are comprised of three types of procedures:

- main instruction or procedural section
- functions (more than one can exist)
- subroutines (more than one can exist)

The availability of a variable is dependent on the procedure the script is executing. The following explains the levels of variable availability in Corel SCRIPT:

- **Global variables** are available anywhere in a running script but they and their values cease to exist when the script stops running. Global variables are created in the main section of a script and cannot be created within a subroutine or a function. However, they can be used in the execution of any subroutine or function. Use the GLOBAL statement to create global variables.
- **Local variables** are available in the procedure in which they are declared. If declared in a subroutine or function, a local variable ceases to exist after the procedure finishes execution and is re-created the next time the subroutine or function is called.
- **Static variables** are declared and assigned values inside a subroutine or a user-defined function and are only available while the script executes that subroutine or user-defined function. In contrast to local variables, static variables retain their values after a subroutine or a function terminates. The retained value can be used by the script the next time the subroutine or function is called.

Note

- You can have variables with the same name in a script but they cannot exist in the same script procedure (main section, functions, subroutines). For example, you can have a variable called ABC in a function and in the main section of a script but you cannot have two ABC variables in the main section of a script.
- Your function and subroutines procedures should be self-contained. A variable required only within a procedure should be a local or static variable. Following this advice can make your procedures more modular, enabling you to copy them to other scripts with limited customization. You should avoid using global variables.
- It's a generally accepted programming convention to put variable declaration statements at the beginning of a script's main section, subroutines, or functions.

{button ,AL(`all_vars;static;dim;global;Script_procedures;',0,"Defaultoverview",)} Related Topics

Explicitly declaring and assigning values to a variable

You can use the Corel SCRIPT DIM statement to explicitly declare a variable. By declaring a variable, you are setting its type and allocating storage space for it in the computer's memory. The following lines declare four variables (**A**, **B**, **C**, and **D**):

```
DIM A% 'declares A as an integer
DIM B$ 'declares B as a string
DIM C AS INTEGER 'declares C as an integer
DIM D AS STRING 'declares D as a string
```

Though **C** and **D** are not declared with suffixes (type-declaration characters), you can still refer to them as **C%** and **D$** in a script. Suffixes are optional but using them is good practice because it makes your scripts easier to read and debug. The variables **A%** and **B$** can also be referred to as **A** and **B**, respectively.

You can also explicitly declare a variable and assign a value at the same time. In the following example, **B** is assigned **A**'s value by using the equal operator (=).

```
DIM A% 'declares A as an integer
A = 3 'assigns the value of A to 3
B& = A 'sets B to a long with A's value
```

In the following example, **B** is explicitly declared as an integer and holds the integer value 10 although **A** times **C** equals 10.8. The variable **B** takes the value 10 because it is declared as an integer and the precision in the expression **A * C** is removed after the expression is calculated.

```
DIM A%   'declares A as an integer
DIM C!   'declares C as a single
A = 3   'assigns the value of 3 to A
C = 3.6   'assigns the value of 3.6 to C
B% = A * C   'B is declared as an integer
```

In the above example, if the last statement read **B = A * C** (missing the %), **B** would equal 10.8 and is **implicitly** declared as a single. Corel SCRIPT assigns the highest precision in cases of operations of mixed numeric data types.

Note

- It's a generally accepted programming convention to put declaration statements at the beginning of a procedure (main section, subroutine , or functions).
- Once a variable has been created, its type cannot change. However, you can convert a variable to another type by creating another variable. See the following statements for more information: CBOL, CCUR, CDAT, CDBL, CINT, CLNG, CSNG, and CSTR.
- Formally declaring variables can make your scripts, especially long and complicated scripts, easier to read and modify.
- Explicitly declared variables that aren't assigned a value hold initial values. See Corel SCRIPT data type summary for each data type's initial value.
- During a script run, the availability of a variable to a script changes. See Variable availability for more details.
- You can assign values to variables without using the LET statement. The two following lines both assign 5 to **abc%**:

```
LET abc% = 5
abc% = 5
```

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

Implicitly declaring and assigning values to a variable

You can implicitly declare a variable and assign a value to it by using the equal sign (=) operator. The variable name is placed on the left side of the equal sign and an expression is placed on the right and Corel SCRIPT determines the most appropriate data type and sets it.

The following examples implicitly declare variables:

```
abc = -400444 ' creates a long
abc = -999999999.1234567890 'creates a double
abc = "This is a string"   'creates a string
abc = (E% <= 100) 'creates a Boolean
```

If you're assigning a number to a variable Corel SCRIPT sets the data type to either a long for whole numbers or a double for non-whole numbers (e.g., 1.5, 0.33333). Corel SCRIPT doesn't set variables to integer or single data types unless the expression on the right side of the equal sign operator uses an integer or a single, respectively. For example:

```
DIM A%   'declares A as an integer
DIM C!   'declares C as a single
A = 3   'assigns the value of 3 to A
C = 3.6   'assigns the value of 3.5 to C
B = A   'assigns A's current value to B
D = C   'assigns C's current value to D
E = A * C
```

In the above example:

- The statement **B = A** assigns **A's** current value to **B** and sets **B** as an integer**.**
- The statement **D = C** assigns **C's** current value to **D** and sets **D** as a single**.**
- The statement **E = A * C** assigns 10.8 to **E** and sets **E** as a single. Corel SCRIPT assigns the highest precision in cases of operations of mixed numeric data types.

Note

- Once a variable has been created, its type cannot change. However, you can convert a variable to another type by creating another variable. See the following statements for more information: CBOL, CCUR, CDAT, CDBL, CINT, CLNG, CSNG, and CSTR.
- Explicitly declaring variables can make your scripts, especially long and complicated scripts, easier to read and modify.
- During a script run, the availability of a variable to a script changes. See Variable availability for more details.
- You can assign values to variables without using the LET statement. The two following lines both assign 5 to **abc%**:

```
LET abc% = 5
abc% = 5
```

- A variable can be assigned a value based on its previous value. In the following example, C is set to its previous value plus 1:

```
C = C + 1
```

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

Corel SCRIPT data type summary

A variable's data type determines the data it can hold.

Data Type	Suffix	Byte Storage	Bits	Range	Initial Value
Numeric Data					
Boolean		1	8	TRUE (-1) or FALSE (0)	FALSE
Integer	**%**	2	16	-32,768 to 32,767 (whole numbers)	0
Long	**&**	4	32	-2,147,483,648 to 2,147,483,647 (whole numbers)	0
Single	**!**	4	32	Negative Values: -3.402823E38 to -1.401298E-45 Positive Values: 1.401298E-45 to 3.402823E38	0
Double	**#**	8	64	Negative Values: -1.79763913486232E308 to -4.94065645841247E-324 Positive Values: 4.94065645841247E-324 to 1.79763913486232E308	0
Date		8	64	1 to 2958465 (as a serial number)	1 (as a serial number)
				31/12/1899 00:00:00.0000 to 31/12/9999 23:59:59.9999 (as a date value)	31/12/1899 00:00:00.0000 (as a date)
				(dd/mm/yyyy hh:mm:ss.ssss)	
Currency	**@**	8	64	-922,337,203,685,477.5808 to 922,337,203,685,477.5807	0.0000
Other Data					
String	**$**	1 (per character)	bytes times 8	0 to 32,765 characters (approximate)	""

Note

- The numeric data types are listed in order of precision from lowest to highest.
- Boolean and Date data types don't use suffixes but can still be explicitly declared using the DIM statement:

```
DIM x AS BOOLEAN
DIM y AS DATE
```

{button ,AL(`all_vars;dim;vars_convert;;;;',0,"Defaultoverview",)} Related Topics

Boolean variable data type

Boolean variables are explicitly declared using the **BOOLEAN** keyword and do not use a suffix. Booleans, which are numeric variables, can only equal TRUE (-1) or FALSE (0). If you set a Boolean variable to anything but -1 or 0, the Boolean variable interprets the value as TRUE and resets to -1. Booleans are used to test conditional expressions such as those used with IF...THEN...ELSE...ENDIF. Of the seven numeric variables in Corel SCRIPT, Booleans are of the lowest precision.

Integer variable data type

Integer variables are explicitly declared using the **%** suffix or the **INTEGER** keyword. Integers, which are numeric variables, range in value from -32,768 to 32,767 and are whole numbers only. Of the seven numeric variables in Corel SCRIPT, integers are of the second-lowest precision.

Long variable data type

Long variables are explicitly declared using the **&** suffix or the **LONG** keyword. Longs, which are numeric variables, range in value from -2,147,483,648 to 2,147,483,647 and are whole numbers only. Of the seven numeric variables in Corel SCRIPT, longs are of the third-lowest precision.

Single variable data type

Single variables are explicitly declared using the **!** suffix or the **SINGLE** keyword. Singles, which are numeric variables, range in value from -3.402823E38 to -1.401298E-45 for negative numbers and 1.401298E-45 to 3.402823E38 for positive numbers. Of the seven numeric variables in Corel SCRIPT, singles are of the fourth-highest precision.

Double variable data type

Double variables are explicitly declared using the # suffix or the **DOUBLE** keyword. Doubles, which are numeric variables, range in value from -1.79763913486232E308 to -4.94065645841247E-324 for negative numbers and 4.94065645841247E-324 to 1.79763913486232E308 for positive numbers. Of the seven numeric variables in Corel SCRIPT, doubles are of the third-highest precision.

When a double variable type is used with a relational operator, it is temporaily recast as a single variable type.

Currency variable data type

Currency variables are explicitly declared using the @ suffix or the **CURRENCY** keyword. Currency variables range in value from -922,337,203,685,477.5808 to 922,337,203,685,477.5807. Of the seven numeric variables in Corel SCRIPT, currency variables are of the second-highest precision. In cases where exactness is important, such as calculations involving finance, money, and fixed-points, you should use currency variables.

Date variable data type

Date variables are explicitly declared using the **DATE** keyword and do not use a suffix. Date variables hold date and time values that range from 1 to 2958465 (as a serial number) or 31/12/1899 00:00:00.0000 to 31/12/9999 23:59:59.9999 (as a date value). You can use dates outside this range but they are not supported by Corel SCRIPT and may lead to errors.A serial value of 1 is equal to 1 day or a 24-hour period.

You can use the CURRDATE function to return your system's date and time.

String variable data type

String variables are explicitly declared using the **$** suffix or the **STRING** keyword. Strings can hold 0 to 32,765 characters. Characters include letters, numbers, punctuation, and spaces. Strings can use any of the 256 ANSI characters for Windows applications (code 0 to 255).

See the Corel SCRIPT Character Map for more information.

Corel SCRIPT type declarations

Data Type Name	Suffix
INTEGER	**%**
LONG	**&**
SINGLE	**!**
DOUBLE	**#**
CURRENCY	**@**
STRING	**$**
BOOLEAN	**None**
DATE	**None**

Corel SCRIPT naming convention

The following rules should be kept in mind when naming variables, constants, functions, subroutines, parameters, and arrays:

- Names can be made up any letters in the alphabet, both lowercase and uppercase, the numbers 0 through 9, and the underscore character (_). The initial character must be a letter or the underscore character and can be followed by any combination of letters, numbers, and the underscore character. The following are all valid names:
  x, X123, Corel_6, TheFunction, a1B2, This_is_a_SUB, ThisIsAConstant, My_2_ARRAY
- Names are not case sensitive. For example, the name **ABC** is the same as **abc**, **Abc**, **aBC**, and so on**.**
- The maximum length of a name is 256 characters.
- All names must be unique in a script file. See Reserved Words for a complete listing of words that can't be used as a name.
- Variable names cannot include spaces.

CBOL

Converts a numeric or string value to a Boolean value.

Syntax
CBOL(x{%|&|!|#|@|$}**)**

Argument	Definition
x{%|&|!|#|@|$}	Any number, numeric variable or constant, or a numeric expression.

Note

- A numeric argument equal to 0 will be converted to FALSE. All others will convert to TRUE (-1).

Example

```
x = 354.43
y = CBOL(x)
```

This example sets **y** to TRUE (-1).

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

CINT

Converts a numeric value to an integer.

Syntax
CINT(x{%|&|!|#|@}**)**

Argument	Definition
x{%|&|!|#|@}	Any number, numeric variable or constant, or a numeric expression.

Note

- The result is an integer, capable of a range of -32,768 to 32,767
- In converting the value to an integer, Corel SCRIPT rounds off the value, rather than truncating it. If the decimal portion of the number is 0.5, CINT rounds to the nearest even number. For example, 8.5 rounds to 8, and 9.5 rounds to 10.

Example

```
x = 354.63
y = CINT(x)
```

This example evaluates **y** to 355.

```
xx = 354.43
yy = CINT(xx)
```

This example evaluates **yy** to 354.

{button ,AL(`Corel_SCRIPT_data_type_summary;vars_convert;int;fix;;',0,"Defaultoverview",)} Related Topics

CLNG

Converts a numeric value to a long integer.

Syntax

CLNG(x{%|&|!|#|@}**)**

Argument	Definition
x{%|&|!|#|@}	Any number, numeric variable or constant, or a numeric expression.

Note

- The result will be a long integer capable of accepting vales in the range -2,147,483,648 to 2,147,483,647.
- In converting the value to a long integer, Corel SCRIPT rounds off the value, rather than truncating it.

Example

```
x = 98765578.43
y = CLNG(x)
```

This example would evaluate **y** to 98765578.

{button ,AL(`Corel_SCRIPT_data_type_summary;vars_convert;int;fix;;',0,"Defaultoverview",)} Related Topics

CSNG

Converts a numeric value to a single.

Syntax

CSNG(x{%|&|!|#|@}**)**

Argument	Definition
x{%|&|!|#|@}	Any number, numeric variable or constant, or a numeric expression.

Note

- The result is a floating-point decimal number, capable of accepting the following values:

Negative Values: -3.402823E38 to -1.401298E-45
Positive Values: 1.401298E-45 to 3.402823E38.

- In converting the value to a single, Corel SCRIPT rounds off the value, rather than truncating it.

Example

```
x = 354.987678
y = CSNG(x)
```

This example would evaluate **y** to 354.9877.

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

CDBL

Converts a numeric value to a double.

Syntax

CDBL(x{%|&|!|#|@}**)**

Argument	Definition
x{%|&|!|#|@}	Any number, numeric variable or constant, or a numeric expression.

Note

- The result will be a double floating-point decimal number, capable of accepting the following values:
Negative Values: -1.79763913486232E308 to -4.94065645841247E-324
Positive Values: 4.94065645841247E-324 to 1.79763913486232E308
- In converting the value to a double, Corel SCRIPT rounds off the value, rather than truncating it.

Example

```
x = 35489097326.43
y = CDBL(x)
```

This example would evaluate **y** to 35489097326.43

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

CCUR

Converts a numeric value to the currency data type.

Syntax
CCUR(x{%|&|!|#|@}**)**

Argument	Definition
x{%|&|!|#|@}	Any number, numeric variable or constant, or a numeric expression.

Note

- This function is capable of accepting the following values:
-922,337,203,685,477.5808 to 922,337,203,685,477.5807
- In converting the value to the currency data type, Corel SCRIPT rounds off the value, rather than truncating it.

Example

```
x = 354.432675434
y = CCUR(x)
```

This example would evaluate **y** to 354.4327.

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

CDAT

Converts a numeric value to a date.

Syntax

CDAT(x{%|&|!|#|@}**)**

Argument	Definition
x{%|&|!|#|@}	Any number, numeric variable or constant, or a numeric expression.

Note

- This function denotes a base date of December 31, 1899 at 12:00:00 A.M. as 1.
- Each additional whole number is 1 additional day.
- Each additional fraction is a portion of a day.
- In converting the value to a date, Corel SCRIPT rounds off the value, rather than truncating it.

Example

```
x = 25.25
y = CDAT(x)
```

This example would evaluate **y** toJanuary 24th, 1900 at 6:00:00 A.M.

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

CSTR

Converts a value to a string of characters.

Syntax
CSTR(x{%|&|!|#|@}**)**

Argument	Definition
x{%|&|!|#|@|$}	Any data type.

Note

- A Boolean value of 0, when cast as a string, will return "FALSE". All other values will return "TRUE".

Example

```
x = 354.43
y = CSTR(x)
```

This example would evaluate **y** to "354.43".

{button ,AL(`Corel_SCRIPT_data_type_summary;vars_convert;STR;;;',0,"Defaultoverview",)} Related Topics

Corel SCRIPT Editor

The Corel SCRIPT Editor is a tool you use to create and edit Corel SCRIPT script files. Since script files are Windows text files, the Corel SCRIPT Editor works like a standard text editor, but also includes features to test, debug, and run script files.

Note

- Unlike script files (or macro files) from other companies, Corel SCRIPT files are text only; there is no compiled binary component in the scripts. Before a script is executed, it is compiled internally into a program file.
- Script files are saved with the extension **.CSC** (for Corel SCRIPT) in the SCRIPTS folder, by default.

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

Corel SCRIPT Editor basics

Like most other Windows text editors, you can use the Corel SCRIPT Editor to insert and delete text. The editor also has cut, copy, and paste features. You can have more than one script file open in the Corel SCRIPT Editor at a time. Each script appears in a separate window. Use the commands on the Window menu in the Editor to arrange the script windows or to switch to a different window.

When you create or edit a script using the Corel SCRIPT Editor, you should keep the following in mind:

- Each line can hold up to 255 characters, including spaces.
- As you reach the right edge of the script window with text, the window scrolls to the right as needed. Text does not ever wrap to the next line.
- Each line in a Corel SCRIPT script file can contain more than one statement or command. Multiple statements on a line are separated with a colon (:). Statements cannot be continued over a line.
- Each line in a Corel SCRIPT script must be followed by a hard return. A hard return is inserted when you press the ENTER key.
- Use tabs, blank lines, and multiple spaces to format your script to make it easier to read. Tabs, blank lines, and multiple spaces are ignored during script playback.
- Use the arrow keys or the mouse to move the insertion point in the script window. Press the CTRL key with the left and right arrow keys to move from word to word.
- An indicator at the bottom of the script window shows the current line number. This can help you find and fix errors.

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

Corel SCRIPT Editor debugging features

When you run a script, it may not always perform the way you expect. A script (or program) that does not work correctly is said to have a "bug" in it. The act of finding and correcting these problems is what is traditionally called "debugging." While some mistakes and typographic errors are often easily spotted by looking carefully at a script, some bugs are harder to find. In order to help you in your search, the Corel SCRIPT Editor includes a full set of debugging tools. You can use the following debugging tools in the Corel SCRIPT Editor:

Running a script
An easy way to see if your script has any errors is to run it. If the script does not contain any errors, it executes to the last line. When you run a script that contains programming errors, playback is aborted at the first instance of an error. The error is noted in the Compiler Output window of the Corel SCRIPT Editor. You can also set an option to check for variables that have not been initialized when you run the script.

Checking script syntax
You can check the syntax of each line in a script without running the script. Common syntax errors include misspelling commands, missing operators, and missing punctuation. If errors are found, error messages appear in the Compiler Output window.

Logic errors are the hardest to find and the only indication of a logic error may be a bad value or an unexpected result. The Corel SCRIPT Editor cannot tell you when a logic error is present so it is up to you to test for and find these problems. To help you with this job, the Corel SCRIPT Editor provides the following tools to help you step through a script more carefully, tracking the values of variables, and following the flow of execution:

Executing individual lines in a script
You can run individual script lines using one of the four commands: Run to Cursor, Step Into, Step Over, and Step Out.

Run to Cursor: Executes the script in the active script window to the position of the insertion point. Since the insertion point acts as a breakpoint, using the Run to Cursor command is similar to using a breakpoint.

Step Into: Executes a script line by line. The Step Into command also steps into functions and subroutines to execute line by line.

Step Over: Executes a script line by line. The Step Over command executes an entire procedure (a function or a subroutine) before stopping.

Step Out: Executes the remaining lines in a function or subroutine and returns and stops at the line after the procedure call.

Using a watch
Stepping through a script to watch the flow of execution can tell you a lot about how a script is performing. It is, however, often just as important to be able to look at the contents of variables as a script runs to see what values they contain. You can monitor a variable's value during script execution with a Quickwatch or the Watch window.

Using Breakpoints
When debugging long or complex scripts, it may be difficult to work with the step commands. To work with a long script, it's better to mark one or more specific lines in the script where execution should stop to let you check how things are going. This is done by setting breakpoints.

Help
If you're having trouble understanding a Corel SCRIPT statement's syntax, you can get reference information from the help file.

{button ,AL(`cse_reference;debugging_scripts;common_programming_errors;;script_errors;;',0,"Defaultoverview",)}
Related Topics

Corel SCRIPT Editor windows

The Corel SCRIPT Editor is comprised of the following windows:

Corel SCRIPT Editor window
The Corel SCRIPT Editor window is the application's main window and holds all the windows noted below. You can set this window to always stay on top of your Windows display. Keeping it on top is useful during debugging sessions.

Script windows
In the Corel SCRIPT Editor, more than one script file can be open. Each script file is opened in a separate script window and its file name is noted in the window border. The script windows can be arranged in a variety of layouts. Each script window contains a compiler output window and a watch window.

Compiler Output window
A script's syntax errors are displayed in the Compiler Output window after it has been run or checked for syntax errors. Double-clicking on an error message's line number in the Compiler Output window sends the insertion point to the line containing the error. The line with the error has the  symbol in its left margin after double-clicking.
The Compiler Output window is displayed below the script window and can be hidden or resized, if you choose.

Watch window
The Watch window is used to monitor the value of variables in the script during a debugging session. Each variable being watched displays its current value and the procedure (Main, a function, or a subroutine) where it is found.

The Watch window is displayed below the script window and can be hidden or resized, if you choose.

{button ,AL(`Trappable_error_codes;Script_programming_errors;cse_reference;ht_add_watch_cse;ht_delete_watch_cse;;;',0,"Defaultoverview",)} **Related Topics**

Starting the editor

To start Corel SCRIPT Editor by using the Run command

1. On the Windows desktop, click Start, Run.
2. In the Open edit box, type the Editor's folder location and SCEDIT.
   For example, C:\COREL60\PROGRAMS\SCEDIT
   Corel applications normally reside in the COREL60\PROGRAMS folder.

Note

- If you forget the location of the Corel SCRIPT Editor, click Browse.
- After the program is started, the Corel SCRIPT Editor button is displayed in the Windows taskbar.

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

To start the Corel SCRIPT Editor from a Corel application

From your Corel application:

- From **CorelCAD 1**: click Tools, Corel SCRIPT Editor.
- From **CorelDRAW 6**: click Tools, Scripts, Corel SCRIPT Editor.
- From **CorelFLOW 3**: click Tools, Corel SCRIPT Editor.
- From **Corel PHOTO-PAINT 6**: click Tools, Scripts, Corel SCRIPT Editor.

Note

- Not every Corel application supports Corel SCRIPT programming and script files. Click for a list of Corel applications that support Corel SCRIPT.

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

To start the Corel SCRIPT Editor from Windows

1. On the Windows desktop, click Start, Programs.
2. Point to the folder that contains the Corel SCRIPT Editor if it does not appear on the main Program menu.
   Corel applications normally reside in the COREL folder.
3. Click Corel SCRIPT Editor (scedit.exe).

Note

- After the program is started, the Corel SCRIPT Editor button is displayed in the Windows taskbar.
- To start Corel SCRIPT Editor from Windows NT, open the group window with the Corel SCRIPT Editor icon and double-click the Editor icon.

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

File Menu

To create a new script with the Corel SCRIPT Editor

- Click File, New. An untitled document window opens.

Note

- You can have multiple document windows opened in the Corel SCRIPT Editor.

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

To open a Corel SCRIPT script

1. Click File, Open.
2. If the Corel SCRIPT script is not in the default folder, chose the drive and folder where the Corel SCRIPT script is stored.
3. Double-click the Corel SCRIPT script you want to open.

Note

- You can use wild cards (* and ?) if you're not sure of the name of the file you want to open. For example, typing **script*.csc** in the File Name box and clicking OK lists all CSC files in the selected folder beginning with **script**. Typing **sc?.csc** in the File Name box and clicking OK lists all CSC files in the selected folder that begin with **sc** and are followed by only one more character.

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

To close a Corel SCRIPT script

- Click File, Close.

Note

- If your changes have not been saved, a confirmation message appears.

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

To save a Corel SCRIPT script

- Click File, Save.

Note

- If you're saving a new Corel SCRIPT script, type a name in the File Name box.
- To save a Corel SCRIPT script with a new name, click File, Save As and type a new name in the File Name box.

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

To print a Corel SCRIPT script

- Click File, Print.

Note

- Click the Setup button from Printer dialog or click File, Print Setup to set the paper size and orientation as specified by the active printer.

Tip

- If your script has lines longer than 80 characters, click File, Print Setup to change page orientation to landscape. The Corel SCRIPT Editor does not automatically wrap long lines when printing.

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

To show how a Corel SCRIPT script will look when printed

- Click File, Print Preview.

Note

- Click File, Print Setup to set the paper size and orientation as specified by the active printer.

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

To close the Corel SCRIPT Editor

- Click File, Exit.

Note

- You are prompted to save any unsaved changes in any open documents.

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

Edit menu

To undo editing operations

- Click Edit, Undo.

Note

- You can't undo editing operations after the script has been saved.

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

To restore changes reversed by the Undo command

- Click Edit, Redo.

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

To copy text to another location

1. Select the text.
2. Click Edit, Copy.
3. Place the insertion point in the document window where you want paste the text.
4. Click Edit, Paste.

Note

- The selected text remains on the Clipboard until you cut or copy another selection to it from any Windows application.

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

To cut text to move to another location

1. Select the text.
2. Click Edit, Cut.
3. Place the insertion point in the document window where you want paste the text.
4. Click Edit, Paste.

Note

- The selected text remains on the Clipboard until you cut or copy another selection to it from any Windows application.

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

To delete text

1. Select the text you want to delete.
2. Click Edit, Delete.

   The selected text is not transferred to the Clipboard.

Note

- You can also delete text without selecting it by pressing the BACKSPACE and DELETE key. The BACKSPACE key deletes text to the left of the insertion point and the DELETE key deletes text to the right of the insertion point.
- Instead of using Edit, Delete, you can delete text by clicking Edit, Cut which transfers controls from the script to the Clipboard.

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

To find text

1. Click in the document where you want to begin searching.
2. Click Search, Find.
3. Enter the text you want to find in the Find What box.
4. Click Find Next.

Tip

- To find and replace text, click Edit, Replace instead of Edit, Find.

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

To find and replace text

1. Click in the document where you want to begin searching.
2. Click Search, Replace.
3. In the Find What box, enter the text you want to find.
4. In the Replace With box, enter the replacement text.
5. Click Replace All to replace all occurrences of the text to find.

Note

- To replace individual text occurrences, click Find Next, Replace instead of clicking Replace All.

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

To place REM statements at the beginning of script lines

1. Place the insertion point in the line where you want to place a REM statement. If you want to place REM statements in a contiguous block of statements, select the statements.
2. Click Edit, Comment.

Note

- Script lines that begin with REM statements are ignored during script execution. This feature can be useful during debugging sessions.
- Use the UnComment command to remove REM statements from selected lines in a script.

{button ,AL(`rem;ht_uncomment;ht_comment;;;',0,"Defaultoverview",)} Related Topics

To remove REM statements from the beginning of script lines

1. Place the insertion point in the line where you want to remove a REM statement. If you want to remove REM statements in a contiguous block of statements, select the statements.
2. Click Edit, UnComment.

Note

- Script lines that begin with REM statements are ignored during script execution. This feature can be useful during debugging sessions.

{button ,AL(`rem;ht_uncomment;ht_comment;;;',0,"Defaultoverview",)} Related Topics

To go to a line in the Corel SCRIPT Editor

1. Click Search, Go to Line.
2. In the Line number box, enter a line number.

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

To go to the next error in a script

- Click Search, Next Error.

Note

- Before running this command, the Compiler Output window must display at least one error message and the insertion point must be in the script window.
- The line where the insertion point is sent has the  symbol displayed in its left margin.

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

To go to the previous error in a script

- Click Search, Previous Error.

Note

- Before running this command, the Compiler Output window must display at least one error message and the insertion point must be in the script window.
- The line where the insertion point is sent has the  symbol displayed in its left margin.

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

To change display colors in the Corel SCRIPT Editor

1. Click Settings, Colors.
2. Click the text color you want in the Text Foreground list box.
3. Click the background color you want in the Text Background list box.

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

To change display and print fonts in the Corel SCRIPT Editor

1. Click Settings, Fonts.
2. Click the font you want in the Font list box.
3. Click the style you want in the Font style list box.
4. Click the size you want in the Size list box.

Note

- You can only use monospaced fonts in the Corel SCRIPT Editor. Monospaced fonts have an equal amount of horizontal space allotted for each character, regardless of its width (an **i** is given as much space as a **w**).

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

To change tab width in the Corel SCRIPT Editor

1. Click Settings, Editor.
2. Enter a number in the Tab Width number box. Tab width is expressed in characters.

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

To set autosave in the Corel SCRIPT Editor

1. Click Settings, Editor.
2. Click the AutoSave checkbox. A checkmark in the checkbox indicates autosave is enabled.
   Click AutoSave again to disable auto save.

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

Debug menu

To run a Corel SCRIPT script from the Corel SCRIPT Editor

- Click Debug, Run.

Note

- If the script you're running contains Corel application commands, the script must use the WITHOBJECT...END WITHOBJECT statements.
- You can also use the procedure above to restart a paused script at the current line.
- To check for unitialized variables, click Option, Uninitialized Variables before Debug, Run. A checkmark indicates the option Uninitialized Variables is enabled.
- You can terminate a script's execution by pressing ESC. In some cases, you may have to press ESC several times to terminate execution.

{button ,AL(`ht_open_file_cse;ht_close_file_cse;ht_debug;;;',0,"Defaultoverview",)} Related Topics

To stop the execution of a script

- Click Debug, Reset.

Note

- You can only use this command when you've paused script execution by stepping, using breakpoints, or when script execution has finished.

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

To interrupt an executing script

- Click ESC.

Note

- After interrupting a script, you can step through it.

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

To restart a script while debugging

- Click Debug, Restart.

Note

- You can only use this command when you've paused script execution by stepping, using breakpoints, pressing ESC, or when script execution has finished.

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

To execute a script one line at a time (Step Into)

To start stepping from the beginning of a script:

- Click Debug, Step Into. Repeat the action for each line you want to execute.

To start stepping from an intermediary position in the script.

1. Click Debug, Toggle Breakpoint where you want to begin stepping.
2. Click Run.

   Script execution is paused at the breakpoint.
3. Click Debug, Step Into. Repeat the action for each line you want to execute.

Note

- The line with the ⇨ symbol in its left margin is the next line to execute.

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

To execute a script one line at a time stepping over procedures (Step Over)

To start stepping from the beginning of a script:

- Click Debug, Step Over. Repeat the action for each line you want to execute.

  When a procedure is encountered, it is executed in its entirety. Execution is then paused after the procedure call.

To start stepping from an intermediary position in the script.

1. Click Debug, Toggle Breakpoint where you want to begin stepping.
2. Click Run.

   Script execution is paused at the breakpoint.
3. Click Debug, Step Over. Repeat the action for each line you want to execute.

   When a procedure is encountered, it is executed. Execution is then paused after the procedure call.

Note

- The line with the ⇨ symbol in its left margin is the next line to execute.

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

To execute a script to the first line after the current procedure call (Step Out)

To start stepping from the beginning of a script:

- Click Debug, Step Out.

  The debugger runs to the first line following the current procedure call and pauses.

To start stepping from an intermediary position in the script.

1. Click Debug, Toggle Breakpoint where you want to begin stepping.
2. Click Run.

   Script execution is paused at the breakpoint.
3. Click Debug, Step Out.

   The debugger runs to the first line following a procedure call and pauses.

Note

- The line with the ⇨ symbol in its left margin is the next line to execute.

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

To add a variable to the Watch window

1. Place the insertion point on a variable in your script.
2. Click Debug, QuickWatch.
3. Click Add Watch

Note

- You can also add a variable to the Watch window by entering it in the Watch window text box.
- Watches are not part of a script. Additionally, they cannot be saved and are lost when you close the script window.

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

To delete a variable from the Watch window

1. Select a watch in the watch window.
2. Click  in the Watch window.

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

To display a variable's value using the QuickWatch

1. Place the insertion point on a variable in your script.
2. Click Debug, QuickWatch.

Note

- You can use this command only when you've paused the script's execution by stepping or using breakpoints.
- Click Cancel to close the QuickWatch window.
- You can type in any variable in the QuickWatch window to return its value.

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

To add or remove a breakpoint

1. Place the insertion point on a line to which you want to add or remove a breakpoint.
2. Click Debug, Toggle Breakpoint.

Note

- Breakpoints cannot be saved and are lost when you close the script window.
- A line with a breakpoint has the  symbol in its left margin.

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

To run a script to the cursor

1. Place the insertion point on the line where you want the script execution to stop.
2. Click Debug, Run To Cursor.

Note

- Since the insertion point acts as a breakpoint, using the Run to Cursor command is similar to using a breakpoint.

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

To clear all breakpoints

- Click Debug, Clear All Breakpoints.

Note

- Breakpoints cannot be saved and are lost when you close the script window.
- A line with a breakpoint has the

 symbol in its left margin.

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

To check a Corel SCRIPT script for syntax errors

- Click Debug, Check Syntax.

Note

- If errors are found, error messages appear in the Compiler Output window. Double-click an error message's line number in the Compiler Output window to send the insertion point to the line containing the error. The line with the error has the
- symbol in its left margin after double-clicking.

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

Window menu

To view all script windows

- Click Window, Arrange All.

Note

- Minimized script windows are arranged at the bottom of the Corel SCRIPT Editor window.

{button ,AL(`ht_windows_cse;ht_open_file_cse;ht_close_file_cse;;;',0,"Defaultoverview",)} Related Topics

To cascade script windows in the Corel SCRIPT Editor

- Click Window, Cascade.

Note

- Minimized script windows are arranged at the bottom of the Corel SCRIPT Editor window.

{button ,AL(`ht_windows_cse;ht_open_file_cse;ht_close_file_cse;;;',0,"Defaultoverview",)} Related Topics

To close all script windows in the Corel SCRIPT Editor

- Click Window, Close All.

Note

- You are prompted to save any unsaved changes in any open documents.

{button ,AL(`ht_windows_cse;ht_open_file_cse;ht_close_file_cse;;;',0,"Defaultoverview",)} Related Topics

To arrange minimized script windows

- Click Window, Arrange Icons

Note

- Minimized dialog editor windows are arranged from the bottom-left corner of the Corel SCRIPT Editor to the bottom-right corner.

{button ,AL(`ht_windows_cse;ht_open_file_cse;ht_close_file_cse;;;',0,"Defaultoverview",)} Related Topics

To keep the Corel SCRIPT Editor window visible

- Click Window, Always on Top. Choose the command again to turn off the setting.

Tip

- Keeping the Corel SCRIPT Editor window visible, even when another application is active, is helpful when you're debugging a script.

{button ,AL(`ht_windows_cse;ht_open_file_cse;ht_close_file_cse;debugging_scripts;;',0,"Defaultoverview",)} Related Topics

To view an open script window in the Corel SCRIPT Editor

- Click Window and click the window you want to view. The open windows are noted at the bottom of the Window menu.

{button ,AL(`ht_windows_cse;ht_open_file_cse;ht_close_file_cse;;;',0,"Defaultoverview",)} Related Topics

To start Help from Corel SCRIPT Editor

- Click Help, Help Topics.

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

To open Corel SCRIPT online Help to a keyword's syntax reference

1. Place the insertion point in the keyword you want help for.
2. Press F1.

Note

- If the selected keyword is not found or the insertion point is not placed in a word, Corel SCRIPT online Help displays the Help Topics dialog box.

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

To view or hide status bar (Corel SCRIPT Editor)

- Click View, Status Bar.

A checkmark beside the Status Bar menu command indicates the status bar is displayed.

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

To view or hide Watch window

- Click View, Watch Window.

A checkmark beside the Watch Window menu command indicates the Watch window is displayed.

Note

- The Watch window can be resized by clicking on a border and dragging.

{button ,AL(`ht_statusbar;ht_watch;ht_compiler;Corel_SCRIPT_Editor_windows;;',0,"Defaultoverview",)} **Related Topics**

To view or hide Compiler Output window

- Click View, Compiler Output Window.

A checkmark beside the Compiler Output Window menu command indicates the Compiler Output window is displayed.

Note

- The Compiler Output window can be resized by clicking on a border and dragging.

{button ,AL(`ht_statusbar;ht_watch;ht_compiler;Corel_SCRIPT_Editor_windows;;',0,"Defaultoverview",)} Related Topics

Corel SCRIPT Editor menu shortcut keys

Press	To
File Menu	
CTRL+N	New
CTRL+O	Open
CTRL+S	Save
CTRL+P	Print
Edit Menu	
CTRL+Z	Undo
ALT+A	Redo
CTRL+X	Cut
CTRL+C	Copy
CTRL+V	Paste
DEL	Delete
CTRL+K	Comment
CTRL+U	UnComment
CTRL+A	Select All
CTRL+D	Dialog
Search Menu	
CTRL+F	Find
CTRL+R	Replace
CTRL+G	Go To Line
F4	Next Error
SHIFT+F4	Previous Error
View Menu	
ALT+1	Watch Window
ALT+2	Compiler Output Window
Debug Menu	
F5	Run
SHIFT+F5	Restart
ALT+F5	Reset
F8	Step Into
F10	Step Over
SHIFT+F7	Step Out
CTRL+W	Quick Watch
F9	Toggle Breakpoint
CTRL+F5	Check Syntax
Help Menu	
F1	Help Topics

F1 help for the menus

File menu

New (File menu, Corel SCRIPT Editor)

Opens an untitled script window.

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

Open (File menu, Corel SCRIPT Editor)

Opens the Run Script dialog box which is used open a saved script file. The default folder and drive are shown but you can open a script file in any drive or folder.

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

Close (File menu, Corel SCRIPT Editor)

Closes the active script window. If your changes have not been saved, a confirmation message appears.

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

Save (File menu, Corel SCRIPT Editor)

Saves the script in the active window. If the script window is untitled, the Save As dialog box appears.

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

Save As (File menu, Corel SCRIPT Editor)

Saves the script in the active window for the first time or saves the script in the active window with a new name.

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

Print (File menu, Corel SCRIPT Editor)

Prints the script in the active window. If the lines in your script are long, use the Print Setup command to set your printer to landscape orientation before printing a script.

{button ,AL(`sce_file;ht_print_file_cse;ht_print_preview_cse;;;',0,"Defaultoverview",)} Related Topics

Print Setup (File menu, Corel SCRIPT Editor)

Displays the Print Setup dialog box, which allows you to choose the printer and printer options.

{button ,AL(`sce_file;ht_print_file_cse;ht_print_preview_cse;;;',0,"Defaultoverview",)} Related Topics

Print Preview (File menu, Corel SCRIPT Editor)

Displays how the script in the active window will look when you print it. You can also print and zoom in print preview mode. If the lines in your script are long, use the Print Setup command on the file menu to set your printer to landscape orientation before printing a script.

{button ,AL(`sce_file;ht_print_file_cse;ht_print_preview_cse;;;',0,"Defaultoverview",)} Related Topics

Exit (File menu, Corel SCRIPT Editor)

Closes all open script windows and the Corel SCRIPT Editor. If you have not saved your scripts, you are prompted to save before exiting.

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

Edit menu

Undo (Edit menu, Corel SCRIPT Editor)

Reverses actions performed during the current session. Use Undo after making a change you do not want implemented. Immediately after choosing Undo, the Redo command becomes available, allowing you to restore what you just undid.

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

Redo (Edit menu, Corel SCRIPT Editor)

Restores changes reversed by the Undo command. Redo becomes available immediately after you choose the Undo command.

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

Cut (Edit menu, Corel SCRIPT Editor)

Cuts selected text from a script and places it on the Clipboard.

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

Copy (Edit menu, Corel SCRIPT Editor)

Copies selected text from a script and places it on the Clipboard.

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

Paste (Edit menu, Corel SCRIPT Editor)

Pastes text from the Clipboard at the insertion point. If you've selected text in a script, it is overwritten with the Clipboard contents.

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

Delete (Edit menu, Corel SCRIPT Editor)

Deletes selected text from a script. If no further action has been performed, you can restore deleted text using the Undo command.

Note

- Instead of deleting text, you can cut it. Cutting text transfers it to the Clipboard.

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

Select All (Edit menu, Corel SCRIPT Editor)

Selects all the text in active script window.

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

Find (Search menu, Corel SCRIPT Editor)

Searches for specified text from the insertion point. You can set the search direction, match the case, and match entire words.

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

Replace (Search menu, Corel SCRIPT Editor)

Searches for and replaces specified text from the insertion point. You can set the search direction, match the case, and match entire words.

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

Go To Line (Search menu, Corel SCRIPT Editor)

Opens a dialog box that lets you to choose the line to go to in a script. The status bar displays line where the insertion point rests.

{button ,AL(`ht_goto_line_cse;sce_debug;next_prev;;',0,"Defaultoverview",)} Related Topics

Next Error (Search menu, Corel SCRIPT Editor)

Sends the insertion point to the next line in a script containing an error. The line where the insertion point is sent has the ▪ symbol displayed in its left margin.

Before running this command, the Compiler Output window must display at least one error message and the insertion point must be in the script window.

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

Previous Error (Search menu, Corel SCRIPT Editor)

Sends the insertion point to the previous line in a script containing an error. The line where the insertion point is sent has the ▪ symbol displayed in its left margin.

Before running this command, the Compiler Output window must display at least one error message and the insertion point must be in the script window.

{button ,AL(`ht_goto_line_cse;sce_edit;sce_debug;next_prev;;',0,"Defaultoverview",)} **Related Topics**

Dialog (Edit menu, Corel SCRIPT Editor)

Opens the Corel SCRIPT Dialog Editor. The Dialog Editor is a tool used to create dialog boxes for scripts.

If the insertion point is in a dialog box definition, the definition is copied to the Corel SCRIPT Dialog Editor and is pasted as a dialog box in a Dialog Editor window. If the insertion point is not in a dialog box definition, a new dialog box is created in the Dialog Editor.

{button ,AL(`htde_move_de_cse;htde_move_cse_cse;sce_edit;htde_move_cse_de;;',0,"Defaultoverview",)} **Related Topics**

View menu

Watch Window (View menu, Corel SCRIPT Editor)

Opens and closes the Watch window. The Watch window monitors the value of specified variable in a script during a debugging session.

The Watch window can be resized by clicking on a border and dragging.

{button ,AL(`Corel_SCRIPT_Editor_windows;sce_view;ht_add_watch_cse;ht_delete_watch_cse;;',0,"Defaultoverview",)}
Related Topics

Compiler Output Window (View menu, Corel SCRIPT Editor)

Opens and closes the compiler output window. Before a script is run, it is compiled into an executable program. If errors during compilation occur, they are displayed in the Compiler Output window. The window updates each time a script is played or checked for syntax errors.

The Compiler Output window can be resized by clicking on a border and dragging.

{button ,AL(`Corel_SCRIPT_Editor_windows;sce_view;ht_add_watch_cse;ht_delete_watch_cse;;',0,"Defaultoverview",)}
Related Topics

ToolBars (View menu, Corel SCRIPT Editor)

Opens a dialog box to display and hide toolbars. You can also use the dialog box to resize buttons.

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

Debug menu

Run (Debug menu, Corel SCRIPT Editor)

Runs the script in the active script window.

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

Restart (Debug menu, Corel SCRIPT Editor)

Stops and restarts script execution from the beginning. Variables are reset to their initial values. You can only use this command when you've paused script execution by stepping or using breakpoints.

{button ,AL(`sce_debug;ht_reset_cse;ht_restart_cse;;;',0,"Defaultoverview",)} Related Topics

Reset (Debug menu, Corel SCRIPT Editor)

Ends script execution and resets variables to their initial values. You can only use this command when you've paused script execution by stepping or using breakpoints.

{button ,AL(`sce_debug;ht_reset_cse;ht_restart_cse;;;',0,"Defaultoverview",)} Related Topics

Step Into (Debug menu, Corel SCRIPT Editor)

Executes a script line by line. The Step Into command also steps into functions and subroutines to execute line by line.

Note

- The line with the ➪ symbol in its left margin is the next line to execute.

{button ,AL(`sce_debug;ht_step_cse;ht_trace_cse;ht_stepout_cse;;',0,"Defaultoverview",)} Related Topics

Step Over (Debug menu, Corel SCRIPT Editor)

Executes a script line by line. The Step Over command executes an entire procedure (a function or a subroutine) without stepping into the procedure's code.

Note

- The line with the ➪ symbol in its left margin is the next line to execute.

{button ,AL(`sce_debug;ht_step_cse;ht_trace_cse;ht_stepout_cse;;',0,"Defaultoverview",)} **Related Topics**

Step Out (Debug menu, Corel SCRIPT Editor)

Executes the remaining lines in a function or subroutine and returns and stops at the line after the procedure call.

Note

- The line with the ⇨ symbol in its left margin is the next line to execute.

{button ,AL(`sce_debug;ht_step_cse;ht_trace_cse;ht_stepout_cse;;',0,"Defaultoverview",)} Related Topics

Run to Cursor (Debug menu, Corel SCRIPT Editor)

Runs the script in the active script window to the position of the insertion point. Since the insertion point acts as a breakpoint, using the Run to Cursor command is similar to using a breakpoint.

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

QuickWatch (Debug menu, Corel SCRIPT Editor)

Opens the QuickWatch dialog box to monitor the value of a variable in the script. You can also use the QuickWatch window to add watches in the Watch Window.

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

Toggle Breakpoint (Debug menu, Corel SCRIPT Editor)

Sets a breakpoint on the line the insertion point is placed on. Choose the command again to clear the breakpoint. Script execution stops at a breakpoint, and you can have more than one breakpoint in a script.

Note

- A line with a breakpoint has the
- symbol in its left margin.

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

Clear All Breakpoints (Debug menu, Corel SCRIPT Editor)

Clears all breakpoints in a script. Script execution stops at a breakpoint, and you can have more than one breakpoint in a script.

Note

- A line with a breakpoint has the
- symbol in its left margin.

{button ,AL(`sce_debug;ht_breakpoint_clear_cse;ht_breakpoint_cse;;;',0,"Defaultoverview",)} Related Topics

Check Syntax (Debug menu, Corel SCRIPT Editor)

Checks for syntax errors in a script. Common syntax errors include misspelling commands, missing operators, and missing punctuation. If errors are found, error messages appear in the Compiler Output window.

Double-click an error message's line number in the Compiler Output window to send the insertion point to the line containing the error. The line with the error has the ▪ symbol in its left margin after double-clicking.

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

Uninitialized Variables (Debug menu, Corel SCRIPT Editor)

An option that you can enable or disable used in conjunction with the Run command. When the Unitialized Variables option is enabled (indicated by a checkmark), the Run command checks for variables that have not been initialized to a value before they are used.

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

Settings menu

Editor (Settings menu, Corel SCRIPT Editor)

Opens a dialog box to set the tab width in characters. You can also set an option to save your script before it's executed by enabling the AutoSave checkbox.

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

Fonts (Settings menu, Corel SCRIPT Editor)

Opens a dialog box to set the font used in the Corel SCRIPT Editor.

Note

- You can only use monospaced fonts in the Corel SCRIPT Editor. Monospaced fonts allot an equal amount of horizontal space for each character, regardless of its width (an **i** is given as much space as a **w**).

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

Colors (Settings menu, Corel SCRIPT Editor)

Opens a dialog box to set text and background colors in the Corel SCRIPT Editor.

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

Window menu

Cascade (Window menu, Corel SCRIPT Editor)

Layers script windows so each title bar is visible. To activate a script window, click the title bar. Minimized windows are arranged at the bottom of the Corel SCRIPT Editor window.

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

Tile (Window menu, Corel SCRIPT Editor)

Arranges the script windows in equal sizes to fit in the Corel SCRIPT Editor. Minimized windows are arranged at the bottom of the Corel SCRIPT Editor window.

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

Always on Top (Window menu, Corel SCRIPT Editor)

Keeps the Corel SCRIPT Editor window visible even when another application is active. This is useful when you are debugging a script. Choose the command again to disable the setting.

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

Help Menu

Help Topics (Help menu, Corel SCRIPT Editor)

Opens the Corel SCRIPT Editor Help Contents screen. From this screen, you can choose the type of Help you want. When you are in Help, clicking on the Contents button takes you back to the opening screen.

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

What's This? command (Help menu)

Changes the cursor to the What's This cursor. When you click a component of the application, a help topic about the object you clicked is displayed.

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

About Corel SCRIPT Editor (Help menu, Corel SCRIPT Editor)

Displays a dialog box with information about the version of Corel SCRIPT Editor you're running. Clicking the System Info button opens the System Info dialog box, which displays information about your system settings.

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

Right-Mouse

Comment (Edit menu, Corel SCRIPT Editor)

Places a REM statement at the beginning of a selected line in a script. Script lines beginning with REM statements are ignored during script execution.

Note

- You can use this feature during debugging sessions to convert script syntax into remarks so they will be ignored during script execution.
- Use the UnComment command to remove REM statements from selected lines in a script.

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

UnComment (Edit menu, Corel SCRIPT Editor)

Removes REM statements from selected lines in a script.

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

Hide (Right-mouse menu, Corel SCRIPT Editor)

Hides the Output Compiler window.

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

Hide (Right-mouse menu, Corel SCRIPT Editor)

Hides the Watch window.

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

What's This? command (Right-mouse menu, Corel SCRIPT Editor)

Opens a What's This? pop-up window containing an explanation of the area you clicked on.

{button ,AL(`hid_help_content;sce_help;cse_help;;;',0,"Defaultoverview",)} Related Topics

Go To Error (Right-mouse menu, Corel SCRIPT Editor)

Sends the insertion point to the line containing the error. The insertion point must first be placed in an error line in the Compiler Output window.

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

Remove Watch (Right-mouse menu, Corel SCRIPT Editor)

Removes the selected variable(s) from the Watch window.

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

Dialog boxes

Go to dialog box

Specifies the line to move the insertion point to in the active script window.

Go To dialog box

Specifies the line to move the insertion point to in the active script window.

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

Quick Watch dialog box

QuickWatch dialog box

The QuickWatch dialog box monitors the value of a variable in the script. You can also use the QuickWatch dialog box to add variables to the Watch window.

Procedure
Displays which procedure in the script the specified variable or expression is located in.

Variable
Displays the variable or expression to watch.

Value
Displays the current value of the specified variable or expression.

Add Watch
Adds the specified variable or expression to the Watch window. Also opens the Watch window, if it is closed.

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

Displays which procedure in the script the specified variable or expression is located in.

Displays the variable or expression to watch.

Displays the current value of the specified variable or expression.

Adds the specified variable or expression to the Watch window. Also opens the Watch window, if it is closed.

Editor dialog box

Specifies the tab spacing to use in scripts. Tabs can help you to format a script so it is easier to read and debug.

Specifies whether scripts should be saved before they are run.

Editor settings dialog box

The Editor settings dialog box sets :

- the default tab width in the Corel SCRIPT Editor
- the option to save your scripts before they are executed.

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

Colors dialog box

List the available text colors.

List the available background colors.

Color settings dialog box

The Color settings dialog box sets the text and background colors in the Corel SCRIPT Editor.

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

OK, Cancel, Help

Opens detailed online Help for this dialog box.

Opens detailed online Help for this dialog box.

Closes this dialog box without saving any changes you have made.

Closes this dialog box and saves any changes you have made.

About dialog box

System Info

Opens the System Info dialog box. Use it to get information about your system, display, network, printing, Corel EXEs & DLLs and system DLLs.

Choose a

Specifies the type of system information you want.

Save

Saves the selected category's details to sysinfo.txt. Once it's saved, a message box appears informing you where the file was saved to.

Screen whats this

HID_BUBBLE_MACEDTYPE_SRVR_IP

HID_BUBBLE_MACEDTYPE_SRVR_EMB

This window displays a Corel SCRIPT script. The window is called the Script window.
Each script line is limited to 255 characters.

HIDC_OUTPUT_EDIT

HIDC_WATCH_PROMPT

Provides a space for to type the name of a variable to add to the Watch Window. Press Enter to add the variable to the Watch Window.

Adds the variable in the Watch box to the Watch window.

Removes the selected variable(s) from the Watch window.

This window displays the current value for each monitored variable and the procedure where the variable is found. The window is called the Watch window.

This window displays errors after a script has been run or checked for syntax errors. The window is called the Compiler Output window.

ToolBars

You can display and hide toolbars in the Corel SCRIPT Editor and the Corel SCRIPT Dialog Editor. You can also resize toolbar buttons.

F1 help for the menus

File Menu

New (File menu, Dialog Editor)

Opens an untitled dialog editor window containing an empty dialog box.

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

Open (File menu, Dialog Editor)

Opens the Open dialog box. Click a saved script file to open. The default folder and drive are shown, but you can open a script file in any drive or folder. If the script contains statements other than dialog box definition statements, it cannot be opened in the Corel SCRIPT Dialog Editor.

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

Close (File menu, Dialog Editor)

Closes the active dialog editor window. If your changes have not been saved, a confirmation message appears.

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

Save (File menu, Dialog Editor)

Saves the dialog box in the active dialog editor window as a Corel SCRIPT script. If the dialog editor window is untitled , the Save As dialog box appears.

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

Save As (File menu, Dialog Editor)

Saves the dialog box in the active dialog editor window as a Corel SCRIPT script for the first time. It also saves the script file with a new name.

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

Open Recently Opened Files (File menu)

Corel SCRIPT Editor

Opens a recently opened Corel SCRIPT script file.

Corel SCRIPT Dialog Editor

Opens a recently opened Corel SCRIPT script file holding a dialog box definition.

{button ,AL(`a_start;c_file_menu;ht_open_file_cse;;;',0,"Defaultoverview",)} Related Topics

Exit (File menu, Dialog Editor)

Closes all open dialog editor windows and the Corel SCRIPT Dialog Editor. If you have not saved, you are prompted to save before exiting.

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

Edit Menu

Undo (Edit menu, Dialog Editor)

Reverses actions performed during the current session. Use Undo after making a change you do not want implemented. Immediately after choosing Undo, the Redo command becomes available, allowing you to restore what you just undid.

rt: a_edit

Redo (Edit menu, Dialog Editor)

Restores changes reversed by the Undo command. Redo becomes available immediately after you choose the Undo command.

Cut (Edit menu, Dialog Editor)

Cuts the selected dialog box, controls, or both from a dialog editor window and places them on the Clipboard as Corel SCRIPT statements.

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

Copy (Edit menu, Dialog Editor)

Copies the selected dialog box, controls, or both from a dialog editor window and places them on the Clipboard as Corel SCRIPT statements.

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

Paste (Edit menu, Dialog Editor)

Pastes Corel SCRIPT dialog control statements into the dialog box in the active dialog editor window from the Clipboard. A pasted control retains the original's label, identifier, size, and position attributes.

Note

- If you paste a control into the same dialog editor window it was copied from, the pasted control is placed on top of the original control with the same name and identifier. You'll have to change the identifier in one of the controls because identifiers must be unique in a dialog definition.
- If you try to paste controls from different-sized dialog boxes into a position that doesn't exist in the second dialog box, the controls are placed in the closest valid position.
- If the controls are too big to fit into the second dialog box, they are resized to fit.
- If you paste Clipboard contents that contain the BEGIN DIALOG and END DIALOG statements, a new dialog box is opened in another dialog editor window.

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

Delete (Edit menu, Dialog Editor)

Deletes selected controls. If no further action has been performed, you can restore a deleted object using the Undo command.

Note

- Instead of deleting a control, you can cut it. Cutting a control transfers it to the clipboard as a Corel SCRIPT statement.

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

Duplicate (Edit menu, Dialog Editor)

Adds a copy of selected control(s) to the active dialog box. By default, the copy is placed on top of the original and offset down and to the right by 3 dialog units.

Note

- The copied control(s) takes on the default control label and identifier. If you copied the control, the original's label and identifier are also copied.

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

Select All (Edit menu, Dialog Editor)

Selects the dialog box in the active dialog editor window and every control in the dialog.

Note

- The Select All command is often used to transfer a dialog box from the Dialog Editor to the Corel SCRIPT Editor.
- The last control you select has a dotted line border.

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

Attributes (Edit menu, Dialog Editor)

Opens the Attributes dialog box for the selected dialog box, controls, or both. You can edit labels, identifiers, dialog and control size, and position attributes from the Attributes dialog box.

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

View Menu

ToolBars (View menu, Dialog Editor)

Opens a dialog box to display and hide toolbars. You can also use the dialog box to resize buttons.

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

Status Bar (View menu)

Corel SCRIPT Dialog Editor

Toggles to display or the status bar found at the bottom of the Corel SCRIPT Editor. The status bar displays:

- menu messages
- the coordinates of the mouse pointer in a dialog box
- the height and width of a selected control or dialog box
- the coordinate position of a selected control or dialog box
- status of NUM lock and CAPS lock

Corel SCRIPT Editor

Toggles to display or hide the status bar found at the bottom of the Corel SCRIPT Editor. The status bar displays:

- menu messages
- status of NUM lock and CAPS lock
- the location of the insertion point in a script window in terms of lines and columns

---

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

Test Dialog (View menu, Dialog Editor)

Sets the dialog box in the active dialog editor window to test mode. In test mode, you can confirm whether a dialog box meets your requirements and functions properly. It is easier to test a dialog within the Dialog Editor than to make it part of a script and test it by running a script. You cannot edit a dialog box in test mode.

In test mode, you can confirm the following dialog box features:

- tab order within the dialog box
- operation of shortcut keys
- option button grouping
- drop-down list or combo box opening

Note

- Press ESC to exit test mode. Pressing any push button or the Close Dialog button ( ) also exits test mode.
- The following controls are filled with place holders in test mode:
  - list boxes
  - drop-down list boxes
  - combo boxes
  - drop-down combo boxes

  The place holders give you a better idea of what the dialog box will look like when it is run.

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

Grid Settings (View menu, Dialog Editor)

Opens the Grid Settings dialog box, where you can set the following options for all dialog editor windows:

- Snap to Grid
- grid spacing
- grid display

Note

- Grid measurements are expressed in dialog units.
- To show the grid, enable Show Grid in the same dialog box.
- Grid spacing settings are set for all dialog editor windows.

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

Control Menu

Selector (Control menu, Dialog Editor)

Sets the mouse pointer to the Selector state. In the Selector state, the mouse pointer appears as  and is used to select, resize, and drag and drop dialog controls. You can also select, move, and resize dialog boxes when in the Selector state. By default, the mouse pointer is in the Selector state. When it's not, it can be reset by clicking

 from the tool bar or clicking Control, Selector from the menu system.

In the Control state, the mouse pointer appears as  and is used to create dialog controls. As soon as you create a tool, the mouse pointer resets to the Selector state. You can only set the mouse pointer to a Control state by clicking any of the dialog controls in the Control tool bar or any control in the Control menu.

Note

The mouse pointer can only appear in a Control state when it is positioned over an active dialog box in a dialog editor window.

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

Text (Control menu, Dialog Editor)

Sets the mouse pointer to the Control state so you can insert a Text control. In the Control state, the mouse pointer appears as  . You can insert a control by clicking or clicking and dragging in a dialog box.

Note

- To insert a control a multiple number of times, hold down CTRL while you select the control. Press ESC to return the mouse back to the Selector state ().

{button ,AL(`a_insert;c_controls;control_text;text;;',0,"Defaultoverview",)} **Related Topics**

Text Box (Control menu, Dialog Editor)

Sets the mouse pointer to the Control state so you can insert a Text Box control. In the Control state, the mouse pointer appears as . You can insert a control by clicking or clicking and dragging in a dialog box.

Note

- To insert a control a multiple number of times, hold down CTRL while you select the control. Press ESC to return the mouse back to the Selector state ().

{button ,AL(`a_insert;c_controls;control_text_box;textbox;;',0,"Defaultoverview",)} **Related Topics**

OK Button (Control menu, Dialog Editor)

Sets the mouse pointer to the Control state to insert an OK Button control. In the Control state, the mouse pointer appears as ▪. You can insert a control by clicking or clicking and dragging in a dialog box.

Note

▪ To insert a control a multiple number of times, hold down CTRL while you select the control. Press ESC to return the mouse back to the Selector state (▪).

{button ,AL(`a_insert;c_controls;control_okbutton;okbutton;;',0,"Defaultoverview",)} **Related Topics**

Cancel Button (Control menu, Dialog Editor)

Sets the mouse pointer to the Control state so you can insert a Cancel Button control. In the Control state, the mouse pointer appears as ▪. You can insert a control by clicking or clicking and dragging in a dialog box.

Note

▪ To insert a control a multiple number of times, hold down CTRL while you select the control. Press ESC to return the mouse back to the Selector state (▪).

{button ,AL(`a_insert;c_controls;control_cancel;cancelbutton;;',0,"Defaultoverview",)} **Related Topics**

Push Button (Control menu, Dialog Editor)

Sets the mouse pointer to the Control state so you can insert a Push Button control. In the Control state the mouse pointer appears as ▪. You can insert a control by clicking or clicking and dragging in a dialog box.

Note

▪ To insert a control a multiple number of times, hold down CTRL while you select the control. Press ESC to return the mouse back to the Selector state (▪).

{button ,AL(`a_insert;c_controls;pushbutton;control_pushbutton;;',0,"Defaultoverview",)} **Related Topics**

Option Button (Control menu, Dialog Editor)

Sets the mouse pointer to the Control state to insert an Option Button control. In the Control state, the mouse pointer appears as ▪. You can insert a control by clicking or clicking and dragging in a dialog box.

Note

▪ To insert a control a multiple number of times, hold down CTRL while you select the control. Press ESC to return the mouse back to the Selector state (▪).

{button ,AL(`a_insert;c_controls;control_option_button;optionbutton;;',0,"Defaultoverview",)} **Related Topics**

Check Box (Control menu, Dialog Editor)

Sets the mouse pointer to the Control state so you can insert a Check Box control. In the Control state, the mouse pointer appears as ▪. You can insert a control by clicking or clicking and dragging in a dialog box.

Note

▪ To insert a control a multiple number of times, hold down CTRL while you select the control. Press ESC to return the mouse back to the Selector state (▪).

{button ,AL(`a_insert;c_controls;checkbox;control_checkbox;;',0,"Defaultoverview",)} **Related Topics**

List Box (Control menu, Dialog Editor)

Sets the mouse pointer to the Control state so you can insert a List Box control. In the Control state, the mouse pointer appears as ▪. You can insert a control by clicking or clicking and dragging in a dialog box.

Note

▪ To insert a control a multiple number of times, hold down CTRL while you select the control. Press ESC to return the mouse back to the Selector state (▪).

{button ,AL(`listbox;control_listbox;a_insert;c_controls;;',0,"Defaultoverview",)} **Related Topics**

Drop-down List Box (Control menu, Dialog Editor)

Sets the mouse pointer to the Control state so you can insert a Drop-down List Box control. In the Control state, the mouse pointer appears as ▪. You can insert a control by clicking or clicking and dragging in a dialog box.

Note

▪ To insert a control a multiple number of times, hold down CTRL while you select the control. Press ESC to return the mouse back to the Selector state (▪).

{button ,AL(`a_insert;c_controls;ddlistbox;control_dropdown_listbox;;',0,"Defaultoverview",)} **Related Topics**

Combo Box (Control menu, Dialog Editor)

Sets the mouse pointer to the Control state so you can insert a Combo Box control. In the Control state, the mouse pointer appears as ▪. You can insert a control by clicking or clicking and dragging in a dialog box.

Note

▪ To insert a control a multiple number of times, hold down CTRL while you select the control. Press ESC to return the mouse back to the Selector state (▪).

{button ,AL(`a_insert;c_controls;combobox;control_combo_box;;',0,"Defaultoverview",)} **Related Topics**

Drop-down Combo Box (Control menu, Dialog Editor)

Sets the mouse pointer to the Control state so you can insert a Drop-down Combo Box control. In the Control state, the mouse pointer appears as ▪. You can insert a control by clicking or clicking and dragging in a dialog box.

Note

▪ To insert a control a multiple number of times, hold down CTRL while you select the control. Press ESC to return the mouse back to the Selector state (▪).

{button ,AL(`a_insert;c_controls;control_dropdown_combobox;ddcombobox;;',0,"Defaultoverview",)} **Related Topics**

Group Box (Control menu, Dialog Editor)

Sets the mouse pointer to the Control state so you can insert a Group Box control. In the Control state, the mouse pointer appears as ▪. You can insert a control by clicking or clicking and dragging in a dialog box.

Note

▪ To insert a control a multiple number of times, hold down CTRL while you select the control. Press ESC to return the mouse back to the Selector state (▪).

{button ,AL(`groupbox;control_group_box;a_insert;c_controls;;',0,"Defaultoverview",)} **Related Topics**

Spin Control (Control menu, Dialog Editor)

Sets the mouse pointer to the Control state so you can insert a Spin control. In the Control state, the mouse pointer appears as ▪. You can insert a control by clicking or clicking and dragging in a dialog box.

Note

▪ To insert a control a multiple number of times, hold down CTRL while you select the control. Press ESC to return the mouse back to the Selector state (▪).

{button ,AL(`control_spin_control;spincontrol;a_insert;c_controls;;',0,"Defaultoverview",)} **Related Topics**

Slider (Control menu, Dialog Editor)

Sets the mouse pointer to the Control state so you can insert a Slider control. In the Control state, the mouse pointer appears as ▪. You can insert a control by clicking or clicking and dragging in a dialog box.

Note

▪ To insert a control a multiple number of times, hold down CTRL while you select the control. Press ESC to return the mouse back to the Selector state (▪).

Horizontal Scroll Bar (Control menu, Dialog Editor)

Sets the mouse pointer to the Control state so you can insert a Horizontal Scroll Bar control. In the Control state, the mouse pointer appears as ▪. You can insert a control by clicking or clicking and dragging in a dialog box.

Note

▪ To insert a control a multiple number of times, hold down CTRL while you select the control. Press ESC to return the mouse back to the Selector state (▪).

Vertical Scroll Bar (Control menu, Dialog Editor)

Sets the mouse pointer to the Control state so you can insert a Vertical Scroll Bar control. In the Control state, the mouse pointer appears as ▪. You can insert a control by clicking or clicking and dragging in a dialog box.

Note

▪ To insert a control a multiple number of times, hold down CTRL while you select the control. Press ESC to return the mouse back to the Selector state (▪).

Progress Indicator (Control menu, Dialog Editor)

Sets the mouse pointer to the Control state so you can insert a Progress Indicator control. In the Control state, the mouse pointer appears as ▪. You can insert a control by clicking or clicking and dragging in a dialog box.

Note

▪ To insert a control a multiple number of times, hold down CTRL while you select the control. Press ESC to return the mouse back to the Selector state (▪).

Image List Box (Control menu, Dialog Editor)

Sets the mouse pointer to the Control state to insert a Image List Box control. In the Control state the mouse pointer appears as . You can insert a control by clicking or clicking and dragging in a dialog box.

Note

To insert a control a multiple number of times, hold down CTRL while you select the control. Press ESC to return the mouse back to the Selector state ( ).

{button ,AL(`imagelist;control_image;a_insert;c_controls;;',0,"Defaultoverview",)} **Related Topics**

Image (Control menu, Dialog Editor)

Sets the mouse pointer to the Control state to insert a Image control. In the Control state the mouse pointer appears as ▪. You can insert a control by clicking or clicking and dragging in a dialog box.

Note

▪ To insert a control a multiple number of times, hold down CTRL while you select the control. Press ESC to return the mouse back to the Selector state (▪).

{button ,AL(`image;control_image;a_insert;c_controls;;',0,"Defaultoverview",)} **Related Topics**

Help Button (Control menu, Dialog Editor)

Sets the mouse pointer to the Control state to insert a Help button control. In the Control state the mouse pointer appears as ▪. You can insert a control by clicking or clicking and dragging in a dialog box.

Note

▪ To insert a control a multiple number of times, hold down CTRL while you select the control. Press ESC to return the mouse back to the Selector state (▪).

{button ,AL(`helpbutton;control_helpbutton;a_insert;c_controls;;',0,"Defaultoverview",)} Related Topics

Arrange Menu

Align Control (Arrange menu, Dialog Editor)

The align commands are used to a arrange selected controls. The controls can be aligned along their left, right, top or bottom edge.

Note

- The last control you select maintains its position; all other selected controls move to align with this control.

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

Align Control, Left (Arrange menu, Dialog Editor)

Aligns selected controls along their left edge. More than one control must be selected to make this command active.

Note

- The last control you select maintains its position; all other selected controls move to align with this control.

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

Align Control, Right (Arrange menu, Dialog Editor)

Aligns selected controls along their right edge. More than one control must be selected to make this command active.

Note

- The last control you select maintains its position; all other selected controls move to align with this control.

{button ,AL(`aligning_and_distributing_dialog_controls;c_align;a_align;;;',0,"Defaultoverview",)} Related Topics

Align Control, Top (Arrange menu, Dialog Editor)

Aligns selected controls along their top edge. More than one control must be selected to make this command active.

Note

- The last control you select maintains its position; all other selected controls move to align with this control.

{button ,AL(`aligning_and_distributing_dialog_controls;c_align;a_align;;;',0,"Defaultoverview",)} Related Topics

Align Control, Bottom (Arrange menu, Dialog Editor)

Aligns selected controls along their bottom edge. More than one control must be selected to make this command active.

Note

- The last control you select maintains its position; all other selected controls move to align with this control.

{button ,AL(`aligning_and_distributing_dialog_controls;c_align;a_align;;;',0,"Defaultoverview",)} Related Topics

Distribute (Arrange menu, Dialog Editor)

The Distribute commands are used to evenly space selected controls across and down a dialog box.

Note

- More than two controls must be selected for this command to be active.

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

Distribute, Horizontal (Arrange menu, Dialog Editor)

Spaces selected controls evenly across a dialog box. The selected controls are spaced evenly between the leftmost and rightmost borders of the selected controls.

Note

- More than two controls must be selected for this command to be active.

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

Distribute, Vertical (Arrange menu, Dialog Editor)

Spaces selected controls evenly down a dialog box. The selected controls are spaced evenly between the topmost and bottommost borders of the selected controls.

Note

- More than two controls must be selected for this command to be active.

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

Center in Dialog (Arrange menu, Dialog Editor)

The Center commands are used to center one or more selected controls in a dialog box, horizontally or vertically.

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

Center in Dialog, Horizontal (Arrange menu, Dialog Editor)

Centers selected control(s) horizontally in a dialog box. The controls are centered horizontally based on the leftmost and rightmost borders of the selected controls.

{button ,AL(`aligning_and_distributing_dialog_controls;c_align;a_align;;;',0,"Defaultoverview",)} Related Topics

Center in Dialog, Vertical (Arrange menu, Dialog Editor)

Centers selected control(s) vertically in a dialog box. The controls are centered vertically based on the topmost and bottommost borders of the selected controls.

{button ,AL(`aligning_and_distributing_dialog_controls;c_align;a_align;;;',0,"Defaultoverview",)} Related Topics

Make Same Size (Arrange menu, Dialog Editor)

The Make Same Size commands are used to make selected controls the same width, height, or both.

{button ,AL(`aligning_and_distributing_dialog_controls;c_align;a_align;;;',0,"Defaultoverview",)} Related Topics

Make Same Size, Width (Arrange menu, Dialog Editor)

Resizes selected controls to make them the same width. The last control you select maintains its width; all other selected controls resize to this control.

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

Make Same Size, Height (Arrange menu, Dialog Editor)

Resizes selected controls to make them the same height. The last control you select maintains its height; all other selected controls resize to this control.

{button ,AL(`aligning_and_distributing_dialog_controls;c_align;a_align;;;',0,"Defaultoverview",)} Related Topics

Make Same Size, Both (Arrange menu, Dialog Editor)

Resizes selected controls to make them the same width and height. The last control you select maintains its width and height; all other selected controls resize to this control.

{button ,AL(`aligning_and_distributing_dialog_controls;c_align;a_align;;;',0,"Defaultoverview",)} Related Topics

Size to Content (Arrange menu, Dialog Editor)

Resizes a control(s) to fit its label.

Note

- You can use this command on option buttons, check boxes, push buttons, and text.

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

Snap to Grid (Arrange menu, Dialog Editor)

Toggles to enable or disable Snap to Grid in all the dialog editor windows. If you're moving and resizing controls in a dialog box using the mouse, enabling Snap to Grid can help to accurately align and position objects along a grid. It also allows you to draw precisely-sized objects.

Note

- Snap to Grid can be enabled without showing the grid.

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

Window Menu

Split (Window menu, Dialog Editor)

The Split command is used to resize the Script window in a dialog editor window. The Script window displays the statements that correspond to the dialog and controls in the dialog editor window.

Though it has no editing functions, the Script window can provide you with a significant amount of information about a dialog box.

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

Cascade (Window menu, Dialog Editor)

Layers dialog editor windows so each title bar is visible. To activate a dialog editor window, click the script's title bar. Minimized windows are arranged at the bottom of the Corel SCRIPT Dialog Editor window.

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

Tile (Window menu, Dialog Editor)

Arranges the dialog editor windows in equal sizes to fit in the Corel SCRIPT Dialog Editor. Minimized windows are arranged at the bottom of the Corel SCRIPT Dialog Editor window.

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

Arrange Icons (Window menu)

Corel SCRIPT Editor

Arranges minimized script windows in the bottom-left corner of the Corel SCRIPT Editor window.

Corel SCRIPT Dialog Editor

Arranges minimized dialog editor windows in the bottom-left corner of the Corel SCRIPT Dialog Editor window.

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

Close All (Window menu, Dialog Editor)

Closes all dialog editor windows. If your changes have not been saved, a confirmation message appears.

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

Open document windows (Window menu)

Corel SCRIPT Editor

Opens and activates a script editor window. The windows are listed in the order in which they were opened.

Corel SCRIPT Dialog Editor

Opens and activates a dialog editor window. The windows are listed in the order in which they were opened.

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

Help Menu

Help Topics (Help menu, Dialog Editor)

Opens the Corel SCRIPT Editor Help Contents screen. From this screen, you can choose the type of Help you want. When you are in Help, clicking on the Contents button takes you back to the opening screen.

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

About Corel SCRIPT Dialog Editor (Help menu, Dialog Editor)

Displays a dialog box with information about the version of Corel SCRIPT Dialog Editor you're running. Clicking the System Info button opens the System Info dialog box, which displays information about your system settings.

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

Dialogs

Attributes dialog box

Provides a space for you to enter the width of the selected dialog box or control(s) in dialog units.

Provides a space for you to enter the height of the selected dialog box or control(s) in dialog units.

Provides a space for you to enter the distance in dialog units from the inside of the dialog box's left border to the left side of the selected control(s).

For dialog boxes, provides a space for you to enter the distance in dialog units from the dialog box's left border to the left side of the monitor's display area.

Provides a space for you to enter the distance in dialog units from the bottom of the dialog box's title bar to the top of the selected control(s).

For dialog boxes, provides a space for you to enter the distance in dialog units from the dialog box's top border to the top side of the monitor's display area.

Centers the dialog box on the monitor during a script run. Ignores the entries in the X and Y boxes when enabled.

Provides a space for you to enter the text for the selected dialog box's title or a control's label.

Provides a space for you to enter a remark for a selected dialog box or control(s).

When the dialog box or control is saved as a Corel SCRIPT statement, the remark is added to the end of the BEGIN DIALOG statement or the control's statement.

Provides a space for you to enter the text for an option group's name. Only active when option buttons are selected.
The option group identifier holds the return value that corresponds to the selected option button within a group.

Provides a space for you to enter the text for a selected dialog box or control's identifier. Identifiers are used to return a value to a running script.

Dialog and Control Attributes dialog box

The following attributes for dialog boxes and controls can be edited in this dialog box:

- Width (in dialog units)
- Height (in dialog units)
- X (left border position)
- Y (top border position)
- Center Dialog option
- Value
- Text
- Comment (a remark is added to the BEGIN DIALOG statement)

The attributes become parameters when the dialog and the controls are saved as Corel SCRIPT statements.

Note

- If the Center Dialog option is enabled when a dialog box is selected, the X and Y number boxes are grayed and cannot be edited. By default, a new dialog has the Center Dialog check box enabled.
- You can select more than one control and open the Attributes dialog box. In this case, the title bar of the Attributes box displays "Multiple Selection Attributes", and allows you to edit all the selected controls at once.

  For example, if you selected a check box and option button, you could set them both to have the same height and width. In some multiple selection cases, attribute options may be grayed and not available to edit.

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

Grid Setting dialog box

The Grid Setting dialog box sets options to help you to accurately align, size, and position controls.

Show Grid

Displays a series of dotted horizontal and vertical lines. Working with the grid on makes it easier to accurately align and position controls.

The dots appear where the lines intersect. The Horizontal and Vertical number boxes set the line spacing.

Snap to Grid

Enabling Snap to Grid forces the mouse pointer to stay on the underlying grid when you insert, move, and resize controls.

Snap to Grid can be enabled without showing the grid.

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

Displays a series of dotted horizontal and vertical lines. Working with the grid on makes it easier to accurately align and position controls. The setting is applied to all dialog editor windows.

The dots appear where the lines intersect. The Horizontal and Vertical number boxes set the line spacing.

Enabling Snap to Grid forces the mouse pointer to stay on the underlying grid when you insert, move, and resize controls. The setting is applied to all dialog editor windows.

You can enable Snap to Grid without showing the grid.

Provides a space for you to enter the grid's horizontal line spacing in dialog units.

Provides a space for you to enter the grid's vertical line spacing in dialog units.

status bar whats this

Displays mouse pointer coordinates in dialog units when positioned over a dialog box in a dialog editor window.
Displays status bar messages when the mouse pointer is not positioned over a dialog box.

Displays the coordinates of the selected control's top-left corner in dialog units. When more than one control is selected, the coordinates of the last selected control is displayed.

Displays the size of the selected control in dialog units. When more than one control is selected, the size of the last selected control is displayed.

dialog editor windows whats this

This is the dialog editor window. The dialog box displayed in the dialog editor window is a graphical representation of Corel SCRIPT statements. Working in this window is similar to using a drawing or painting application: dialog controls are graphic objects which can be inserted, moved, resized, and aligned in a dialog box.

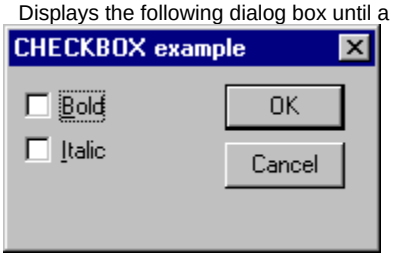
This is the Script window. It displays the Corel SCRIPT statements that correspond to the dialog box and controls in the dialog editor window. The order in which controls are entered into a dialog box determines the order of the dialog box script statements.

You can copy dialog box definition statements from the Script window to the clipboard. However, you cannot perform any editing function in the Script window.

Check box example

```
' Check box example
BEGIN DIALOG Dialogbox1 122, 55, "CHECKBOX example"
    CHECKBOX 5, 7, 60, 10, "&Bold", bold%
    CHECKBOX 5, 20, 60, 10, "&Italic", ital%
    OKBUTTON 72, 5, 40, 14
    CANCELBUTTON 72, 23, 40, 14
END DIALOG
ret = DIALOG(Dialogbox1)
    ' If ret is 2, then Cancel button was chosen
IF ret = 2 THEN STOP

REM Displays a MESSAGE based on selections
IF bold=1 and ital=1 then
    MESSAGE "Bold on; Italic on"
ELSEIF bold=1 and ital=0 then
    MESSAGE "Bold on; Italic off"
ELSEIF bold=0 and ital=1 then
    MESSAGE "Bold off; Italic on"
ELSE
    MESSAGE "Bold off; Italic off"
ENDIF
```

Displays the following dialog box until a OK or CANCEL selected.

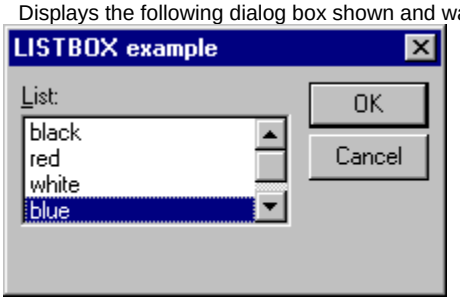


{button ,AL(`dialog_example_all;;;;;',0,"Defaultoverview",)} Related Topics

List box example

```
' List box example
REM Create an array called arr
DIM arr$(5)
arr(1) = "black"
arr(2) = "red"
arr(3) = "white"
arr(4) = "blue"
arr(5) = "green"
index = 4
BEGIN DIALOG listboxdlg 144, 68, "LISTBOX example"
    TEXT 4, 4, 90, 8, "&List:"
    LISTBOX 4, 14, 90, 50, arr, index
    OKBUTTON 100, 4, 40, 14
    CANCELBUTTON 100, 20, 40, 14
END DIALOG
ret = DIALOG(listboxdlg)
' If Cancel is selected, stop the script
IF ret = 2 THEN STOP
MESSAGE "You chose " + arr(index)
```

Displays the following dialog box shown and waits for the user to select an element in the list box and then choose OK.

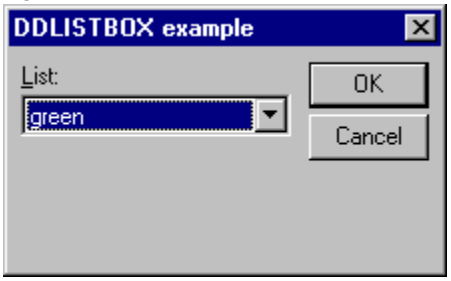


{button ,AL(`dialog_example_all;;;;;',0,"Defaultoverview",)} Related Topics

Drop-down list box example

```
' Drop-down list box example
REM Create an array called arr
DIM arr$(5)
arr(1) = "black"
arr(2) = "red"
arr(3) = "white"
arr(4) = "blue"
arr(5) = "green"
index% = 4'"green"
BEGIN DIALOG DDlistboxdlg 144, 68, "DDLISTBOX example"
    TEXT 4, 4, 90, 8, "&List:"
    DDLISTBOX 4, 14, 90, 50, arr, index
    OKBUTTON 100, 4, 40, 14
    CANCELBUTTON 100, 20, 40, 14
END DIALOG
ret = DIALOG(DDlistboxdlg)
' If Cancel is selected, stop the script
IF ret = 2 THEN STOP
MESSAGE "You chose " + arr(index)
```

Displays the following dialog box shown and waits for the user to select an element in the drop-down list box and then choose OK.

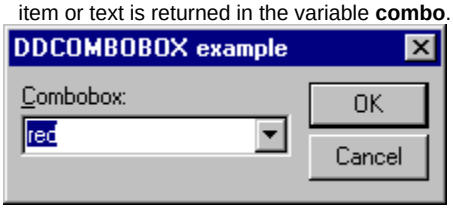


{button ,AL(`dialog_example_all;;;;;',0,"Defaultoverview",)} Related Topics

Drop-down combo box example

```
' Drop-down combo box example
REM Create an array called arr
DIM arr$(5)
arr(1) = "black"
arr(2) = "red"
arr(3) = "white"
arr(4) = "blue"
arr(5) = "green"
combo = arr(2) 'initializes a choice in the combobox
BEGIN DIALOG ddcombodlg 144, 40, "DDCOMBOBOX example"
    TEXT 4, 4, 90, 8, "&Combobox:"
    DDCOMBOBOX 4, 14, 90, 62, arr, combo
    OKBUTTON 100, 4, 40, 14
    CANCELBUTTON 100, 20, 40, 14
END DIALOG
ret = DIALOG(ddcombodlg)
' If Cancel is selected, stop the script
IF ret = 2 THEN STOP

FOR i% = 1 TO 5

' If we are within the bounds of the array and
    ' find a match, then display a message.
    IF combo = arr(i) THEN
        MESSAGE "You chose " + combo
        GOTO ENDFOR
    ENDIF
NEXT i
REM If a match is found, this line will be skipped.
MESSAGE "You chose your own color: " + combo
ENDFOR:
```

Displays the dialog box until the user selects an item from the list or enters text into the text box and selects OK. The selected item or text is returned in the variable **combo**.

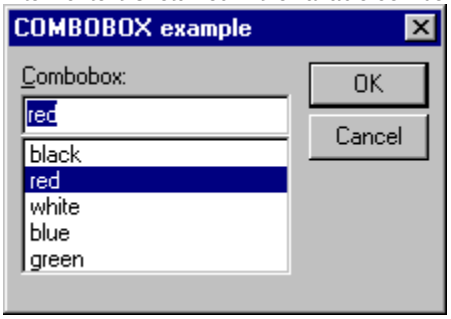


{button ,AL(`dialog_example_all;;;;;',0,"Defaultoverview",)} **Related Topics**

Combo box example

```
' Combo box example
REM Create an array called arr
DIM arr$(5)
arr(1) = "black"
arr(2) = "red"
arr(3) = "white"
arr(4) = "blue"
arr(5) = "green"
combo = arr(2) 'initializes a choice in the combobox
BEGIN DIALOG combodlg 144, 80, "COMBOBOX example"
    TEXT 4, 4, 90, 8, "&Combobox:"
    COMBOBOX 4, 14, 90, 62, arr, combo
    OKBUTTON 100, 4, 40, 14
    CANCELBUTTON 100, 20, 40, 14
END DIALOG
ret = DIALOG(combodlg)
' If Cancel is selected, stop the script
IF ret = 2 THEN STOP

FOR i% = 1 TO 5

' If we are within the bounds of the array and
    ' find a match, then display a message.
    IF combo = arr(i) THEN
        MESSAGE "You chose " + combo
        STOP
    ENDIF
NEXT i
REM If a match is found, this line will be skipped.
MESSAGE "You chose your own color: " + combo
```

Displays the dialog box until the user selects an item from the list or enters text into the text box and selects OK. The selected item or text is returned in the variable **combo**.

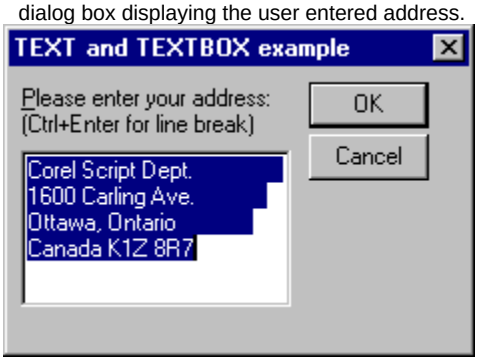


{button ,AL(`dialog_example_all;;;;;',0,"Defaultoverview",)} **Related Topics**

Text and text box example

```
' Text and text box example
text1 = "Corel Script Dept."+CHR(13)+CHR(10)+"1600 Carling Ave."+CHR(13)+CHR(10)+"Ottawa,
Ontario"+CHR(13)+CHR(10)+"Canada K1Z 8R7"
label1 = "&Please enter your address:" + CHR(13) + "(CTRL+ENTER for line break)"
' Initialize dialog box variables
'index = 2
BEGIN DIALOG textdlg 153, 87, "TEXT and TEXTBOX example"
    TEXT  4, 5, 100, 16, label1
    TEXTBOX 4, 25, 90, 48, text1
    OKBUTTON 100, 4, 40, 14
    CANCELBUTTON 100, 20, 40, 14
END DIALOG
ret = DIALOG(textdlg)
' If Cancel is selected, stop the script
IF ret = 2 THEN STOP
MESSAGE "Your Address is: " + CHR(13) + text1
```

Displays the dialog box shown below. In the text box, use CTRL+ENTER for line breaks. The dialog box returns a message dialog box displaying the user entered address.



Note

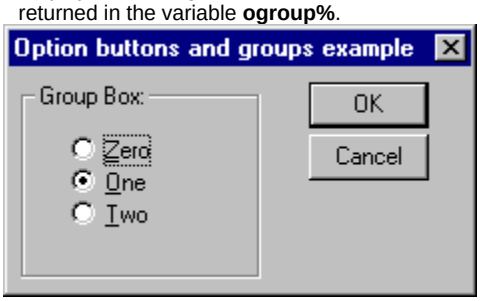
- If you want to insert a line break into the default text for a text box, use ANSI characters 10 and 13. See CHR for more information.

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

Option button, option group, and group box example

```
' Option button, option group, and group box example
ogroup%=1
BEGIN DIALOG OptionDialog 154, 68, "Option buttons and groups example"
    GROUPBOX 4, 4, 80, 60, "Group Box:"
    OPTIONGROUP ogroup%
        OPTIONBUTTON 20, 20, 30, 10, "&Zero"
        OPTIONBUTTON 20, 30, 30, 10, "&One"
        OPTIONBUTTON 20, 40, 30, 10, "&Two"
    OKBUTTON 100, 4, 40, 14
    CANCELBUTTON 100, 20, 40, 14
END DIALOG
return = DIALOG(Optiondialog)
' If return is 2, then Cancel button was chosen
IF return = 2 THEN STOP
SELECT CASE ogroup
    CASE 0
        ' The Zero option button was selected
        MESSAGE "You chose Zero"
    CASE 1
        ' The One option button was selected
        MESSAGE "You chose One"
    CASE 2
        ' The Two option button was selected
        MESSAGE "You chose Two"
END SELECT
```

Displays the dialog box shown below until the user selects one of the options buttons and OK. The response value is then returned in the variable **ogroup%**.

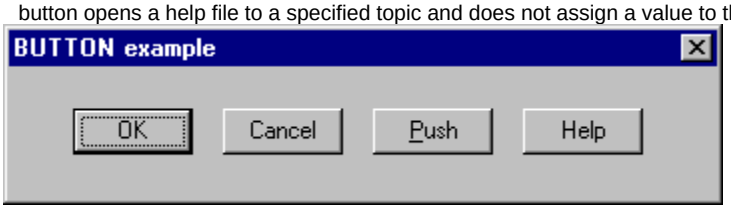


{button ,AL(`dialog_example_all;;;;;',0,"Defaultoverview",)} Related Topics

OK, Cancel, Help, and Push button example

```
' Push button, OK button, Cancel button, and Help button example
BEGIN DIALOG Buttons1 55, 34, 236, 40, "BUTTON example"
    OKBUTTON 21, 12, 40, 14
    CANCELBUTTON 71, 12, 40, 14
    PUSHBUTTON 121, 12, 40, 14, "&Push"
    HELPBUTTON 171, 12, 40, 14, "C:\yourhelp\help.hlp", 104
END DIALOG
ret = DIALOG(Buttons1)
' If ret is 1, then OK button was chosen
IF ret = 1 THEN MESSAGE "OK button chosen"
' If ret is 2, then Cancel button was chosen
IF ret = 2 THEN
    MESSAGE "CANCEL button was chosen"
    STOP
END IF
' If ret is 3, then Push button was chosen
IF ret = 3 THEN MESSAGE "Push button chosen"
```

Displays the dialog box shown below until the user selects either the OK button, Cancel Button, or the Push button. The help button opens a help file to a specified topic and does not assign a value to the **ret** variable



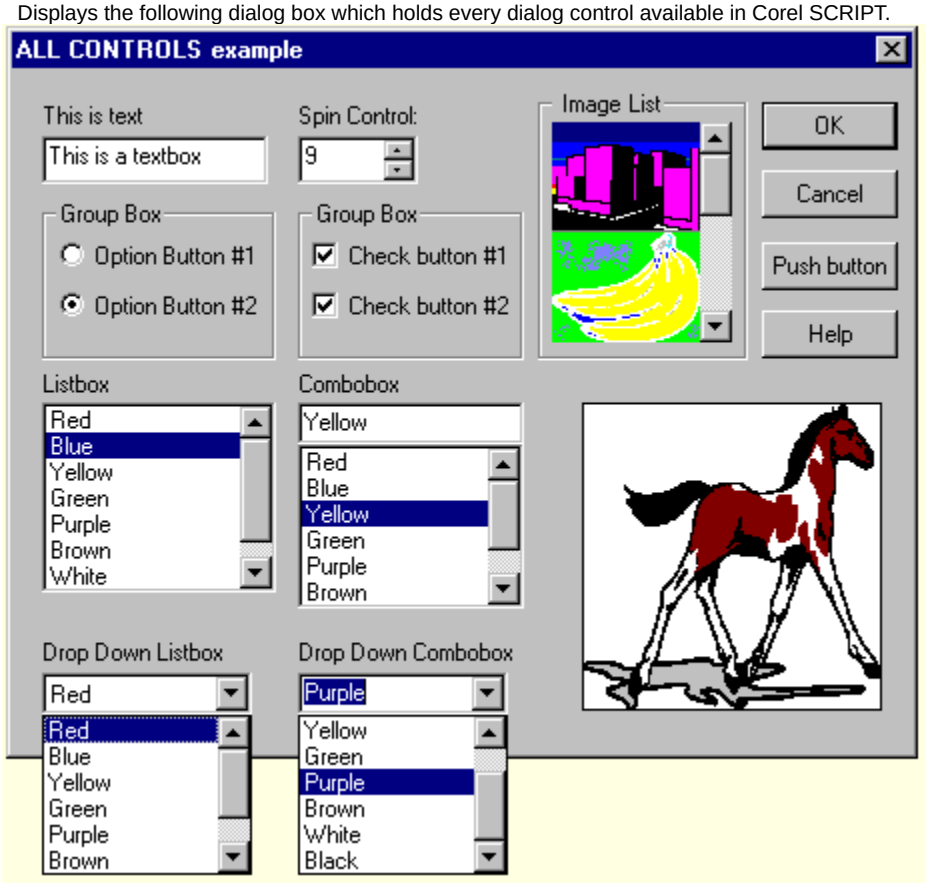
{button ,AL(`dialog_example_all;;;;;',0,"Defaultoverview",)} Related Topics

Example of all dialog controls

```
' Example showing every dialog control used in Corel SCRIPT
DIM bmparray$(5)
DIM color(8) as string
bmpname = "C:\corel60\photopnt\plgbrush\footprnt.bmp"
bmparray(1) = "C:\corel60\photopnt\plgbrush\fuzzy.bmp"
bmparray(2) = "C:\corel60\photopnt\plgbrush\boxpanel.bmp"
bmparray(3) = "C:\corel60\photopnt\plgbrush\textart.bmp"
bmparray(4) = "C:\corel60\photopnt\plgbrush\treebare.bmp"
bmparray(5) = "C:\corel60\photopnt\plgbrush\saturn.bmp"
color(1) = "Red"
color(2) = "Blue"
color(3) = "Yellow"
color(4) = "Green"
color(5) = "Purple"
color(6) = "Brown"
color(7) = "White"
color(8) = "Black"
'**********************************Dialog declaration
text1 = "This is text"
help = "HELP!"
push1 = "Push button"
BEGIN DIALOG d1 300, 210, "ALL CONTROLS example"
    OKBUTTON 250, 10, 45, 15
    CANCELBUTTON 250, 31, 45, 15
    HELPBUTTON 250, 53, 45, 15, help, HelpID%
    PUSHBUTTON 250, 74, 45, 15, push1
    TEXT 10, 10, 80, 15, text1
    TEXTBOX 10, 20, 75, 15, textbox1$
    GROUPBOX 10, 40, 78, 49, "Group Box"
        OPTIONGROUP group1%
            OPTIONBUTTON 15, 50, 66, 15, "Option Button #1"
            OPTIONBUTTON 15, 65, 66, 15, "Option Button #2"
    GROUPBOX 95, 40, 75, 49, "Group Box"
        CHECKBOX 100, 50, 65, 15, "Check Button #1", binary1%
        CHECKBOX 100, 65, 65, 15, "Check Button #2", binary2%
    TEXT 10, 92, 70, 15, "Listbox"
    LISTBOX 10, 102, 78, 70, Color, listbox1%
    TEXT 95, 92, 70, 15, "Combobox"
    COMBOBOX 95, 102, 75, 70, Color, combobox1$
    TEXT 10, 175, 70, 15, "Drop Down Listbox"
    DDLISTBOX 10, 185, 70, 70, Color, ddlistbox1%
    TEXT 95, 175, 75, 15, "Drop Down Combobox"
    DDCOMBOBOX 95, 185, 70, 70, Color, ddcombobox1$
    TEXT 95, 10, 80, 15, "Spin Control: "
    SPINCONTROL 95, 20, 40, 15, spin1%
    IMAGE 190, 102, 100, 95, Bmpname
    GROUPBOX 175, 6, 70, 83, "Picture List"
        IMAGELISTBOX 180, 16, 60, 100, bmparray, imglst%
END DIALOG
'**************************Set defaults
textbox1 = "This is a textbox"
group1 = 1
binary1 = -1
binary2 = 1
binary3 = 0
Listbox1 = 2
Combobox1 = color(3)
DDListBox1 = 1
DDCombobox1 = color(5)
spin1 = 9
imglst = 2
TRYAGAIN:
ret = DIALOG(d1)
if ret = 2 then stop
IF ret = 3 then
    MESSAGE "You pressed a push button"
    GOTO TRYAGAIN
END IF
mess$ = "You entered " + CHR(34) + textbox1 + CHR(34) + " in the textbox"
```

```
mess = mess + CHR(13) + "You picked " + CSTR(spin1) + " in the spin box"
mess$ = mess + CHR(13) + "You selected Option Button #" + CSTR(group1 + 1)
mess = mess + CHR(13) + "You selected:"
if binary1<>0 then mess = mess + " Check Button #1"
if binary2<>0 then mess = mess + " Check Button #2"
mess = mess + CHR(13) + "You picked " + color(Listbox1) + " in the ListBox"
mess = mess + CHR(13) + "You entered " + Combobox1 + " in the ComboBox"
mess = mess + CHR(13) + "You entered " + DDCombobox1 + " in the DDComboBox"
mess = mess + CHR(13) + "You picked " + color(DDListbox1) + " in the DDListBox"
mess = mess + CHR(13) + "You picked " + bmparray(imglst) + " as your image"
MESSAGE mess
```

Displays the following dialog box which holds every dialog control available in Corel SCRIPT.

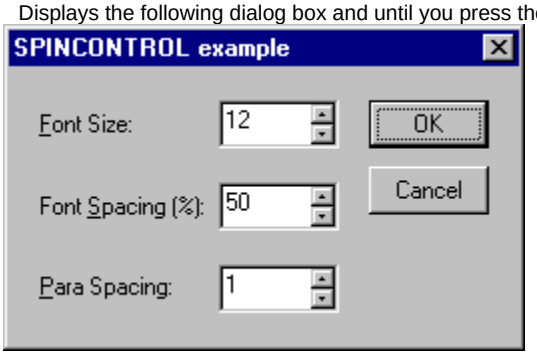


{button ,AL(`dialog_example_all;;;;;',0,"Defaultoverview",)} Related Topics

Spin control example

```
' Spin control example
REM set default values
spin1% = 12
spin2% = 50
spin3% = 1

BEGIN DIALOG spindlg 170, 85, "SPINCONTROL example"
    TEXT 10, 13, 55, 15, "&Font Size: "
         SPINCONTROL 70, 10, 40, 15, spin1%
    TEXT 10, 38, 55, 15, "Font &Spacing (%): "
         SPINCONTROL 70, 35, 40, 15, spin2%
    TEXT 10, 63, 55, 15, "&Para Spacing: "
         SPINCONTROL 70, 60, 40, 15, spin3%
    OKBUTTON 120, 10, 40, 15
    CANCELBUTTON 120, 30, 40, 15
END DIALOG

return = DIALOG(spindlg)
IF return = 2 THEN STOP

MESSAGE "You chose a font size of: " + CSTR(spin1)
MESSAGE "You chose a font spacing of: " + CSTR(spin2) + "%"
MESSAGE "You chose a para spacing of: " + CSTR(spin3)
```

Displays the following dialog box and until you press the OK or Cancel button.

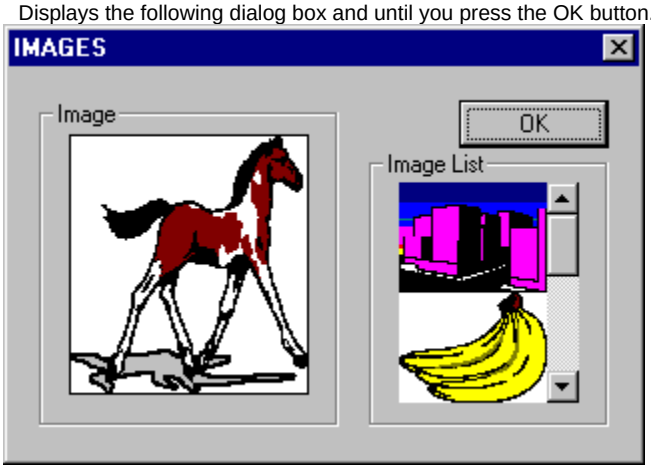


{button ,AL(`dialog_example_all;;;;;',0,"Defaultoverview",)} **Related Topics**

Image list box and Image example

```
' Image list box and Image example
DIM bmparray$(5)
bmpname = "C:\corel60\photopnt\plgbrush\pony.bmp"
bmparray(1) = "C:\corel60\photopnt\plgbrush\building.bmp"
bmparray(2) = "C:\corel60\photopnt\plgbrush\banana.bmp"
bmparray(3) = "C:\corel60\photopnt\plgbrush\textart.bmp"
bmparray(4) = "C:\corel60\photopnt\plgbrush\treebare.bmp"
bmparray(5) = "C:\corel60\photopnt\plgbrush\saturn.bmp"
BEGIN DIALOG D1 210, 120, "IMAGES"
    OKBUTTON 150, 10, 50, 15
    GROUPBOX 10, 10, 100, 100, "Image"
    IMAGE 20, 20, 80, 80, Bmpname
    GROUPBOX 120, 25, 80, 85, "Image List"
        IMAGELISTBOX 130, 35, 60, 100, bmparray, imglst%
END DIALOG
ret% = DIALOG(D1)
```

Displays the following dialog box and until you press the OK button.



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

To start Corel SCRIPT Dialog Editor from the Windows desktop

1. On the Windows desktop, click Start, Programs.
2. Point to the folder that contains the Corel SCRIPT Dialog Editor if it does not appear on the main Program menu. Corel applications normally reside in the COREL folder.
3. Click Corel SCRIPT Dialog Editor (dlged.exe).

Note

- After the program is started, the Corel SCRIPT Dialog Editor button is displayed in the Windows taskbar.
- To start Corel SCRIPT Dialog Editor from Windows NT, open the group window with the Corel SCRIPT Dialog Editor icon and double-click the Dialog Editor icon.

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

To start Corel SCRIPT Dialog Editor by using the Run command

1. On the Windows desktop, click Start, Run.
2. In the Open edit box, type the Dialog Editor's folder location and DLGED.
   For example, C:\COREL60\PROGRAMS\DLGED
   Corel applications normally reside in the COREL60\PROGRAMS folder.

Note

- If you forget the location of the Corel SCRIPT Dialog Editor, click Browse.
- After the program is started, the Corel SCRIPT Dialog Editor button is displayed in the Windows taskbar.

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

To start the Corel SCRIPT Dialog Editor from the Corel SCRIPT Editor

From the Corel SCRIPT Editor:

- Click Edit, Dialog.

If the insertion point is placed in a dialog box definition in the Corel SCRIPT Editor, the dialog box is loaded into a dialog editor window. A dialog box definition consists of the BEGIN DIALOG and END DIALOG statements with dialog control statements in between. The Corel SCRIPT Dialog Editor ignores statements that are not part of the dialog box definition.

If the insertion point is not placed in a dialog box definition in the Corel SCRIPT Editor, an empty dialog box is opened in the Dialog Editor.

Note

- After the program is started, the Corel SCRIPT Dialog Editor button is displayed in the Windows taskbar.

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

To create a new dialog box

- Click File, New.

A dialog editor window opens with an empty dialog box in it.

Note

- Only one dialog box can exist in a dialog editor window but you can have multiple dialog editor windows opened in the Corel SCRIPT Dialog Editor.
- By default, a new dialog has the Center Dialog checkbox enabled in its dialog attributes box.

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

To open a Corel SCRIPT script that contains a dialog box definition

1. Click File, Open.
2. If the Corel SCRIPT script is not in the default folder, choose the drive and folder where it is stored.
3. Double-click the Corel SCRIPT script you want to open.

Note

- If the script contains statements other than dialog box definition statements, it cannot be opened in the Corel SCRIPT Dialog Editor.
- The Dialog Editor can read REM statements but they are ignored and lost when the dialog box is saved.
- You can use wild cards (* and ?) if you're not sure of the name of the file you want to open. For example, typing **script*.csc** in the File Name box and clicking OK lists all CSC files in the selected folder beginning with **script**. Typing **script?.csc** in the File Name box and clicking OK lists all CSC files in the selected folder that begin with **script** and are followed by only one more character.

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

To close a Corel SCRIPT Dialog Editor window

- Click File, Close.

Note

- If your changes have not been saved, a confirmation message appears.

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

To save a Corel SCRIPT dialog box

- Click File, Save.

Note

- Dialog boxes are saved as Corel SCRIPT statements. The statements form a dialog box definition which starts with the BEGIN DIALOG statement, ends with the END DIALOG statement, and has dialog control statements in between.
- If you're saving a new Corel SCRIPT dialog box, type a name in the File Name box.
- To save a Corel SCRIPT dialog box script with a new name, click File, Save As and type a new name in the File Name box.

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

To close the Corel SCRIPT Dialog Editor

- Click File, Exit.

Note

- You are prompted to save any unsaved changes in any open dialog editor windows.

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

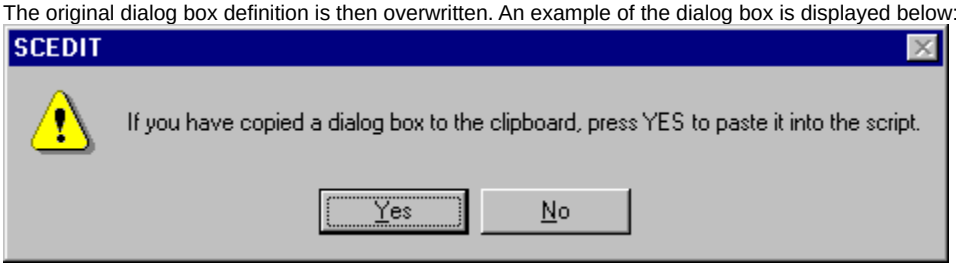
To move or copy a dialog box definition from the Dialog Editor to the Corel SCRIPT Editor

From the Corel SCRIPT Dialog Editor:

1. Click Edit, Select All.
2. Click Edit, Cut to move or Edit, Copy to copy.
   The dialog box definition is transferred to the clipboard as Corel SCRIPT statements consisting of the BEGIN DIALOG and END DIALOG statements with the control statements in between.
3. In the Corel SCRIPT Editor, place the insertion point where you want to insert the dialog box definition.
4. Click Edit, Paste. Selected text in the Editor is overwritten with the clipboard contents.

Note

- You can also transfer control statements from the Dialog Editor to the Editor. Cut or copy the selected controls to the clipboard and then paste them in the Editor.
- If you had brought a dialog box into the Dialog Editor through the Corel SCRIPT Editor by clicking Edit, Dialog, you are prompted to transfer the dialog box definition to the Clipboard when you close the Dialog Editor. You can then paste the transferred dialog box definition to the Editor by clicking Yes in SCEDIT system dialog box that was opened when the Dialog editor was started. The original dialog box definition is then overwritten. An example of the dialog box is displayed below:



{button ,AL(`a_trans;a_insert;a_start;ht_start_cse_app;ht_start_cse_win;;',0,"Defaultoverview",)} Related Topics

To move or copy a dialog box definition from the Corel SCRIPT Editor to the Dialog Editor

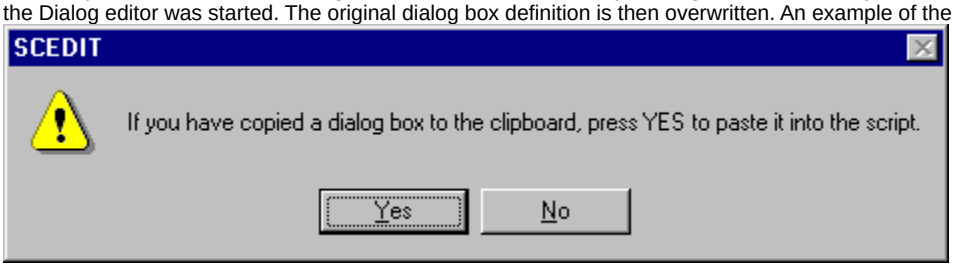
From the Corel SCRIPT Editor:

1. Place the insertion point in a dialog box definition.
2. Click Edit, Dialog.

   If the insertion point is not placed in a dialog box definition in the Corel SCRIPT Editor, an empty dialog box is opened in the Dialog Editor.

Note

- When you close the Corel SCRIPT Dialog Editor, you are prompted to transfer the dialog box definition to the Clipboard. You can paste the transferred dialog box definition to the Editor by clicking Yes in SCEDIT system dialog box that was opened when the Dialog editor was started. The original dialog box definition is then overwritten. An example of the dialog box is displayed below:



{button ,AL(`a_trans;a_insert;a_start;ht_start_cse_app;ht_start_cse_win;;',0,"Defaultoverview",)} Related Topics

To edit a dialog box's attributes

1. Select the dialog box.
2. Click Edit, Attributes.
3. Type a new values into any of the following attribute boxes:

- Width (in dialog units)
- Height (in dialog units)
- X (left border position)
- Y (top border position)
- Center Dialog option
- Value
- Text
- Comment (a remark is added to the BEGIN DIALOG statement).

Note

- If the Center Dialog option is enabled, the X and Y number boxes are grayed and cannot be edited. By default, a new dialog has the Center Dialog checkbox enabled.
- A thick border around a dialog box indicates it has been selected. Dialog boxes don't use sizing handles.

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

To resize a dialog box using the mouse

1. Place the mouse pointer on a dialog border.
2. Drag the side, top or bottom border to resize the window in one direction or drag a border corner to resize the window horizontally and vertically.
3. Release the mouse button when the window is the desired size.

Note

- Pressing ESC while resizing resets the dialog box to its original size.

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

To move a dialog box using the mouse

1. Position the mouse pointer on the dialog box title bar.
2. Hold the mouse button down and drag the dialog box to a new position.

Note

- The dialog editor window is a representation of your computer screen. When you move the dialog box within the dialog editor window you are actually changing the dialog box's screen placement when it is run in a script. For example, if you move the dialog box to the bottom-right corner of its dialog editor window, it will appear on the bottom-right corner of the computer screen when run in a script file.
- If the Center Dialog checkbox in the dialog attributes box is enabled, the dialog box position in the dialog editor window is ignored and the dialog box is centered during script running. A dialog box can also be set to be centered on the screen by omitting the position attributes parameters in the BEGIN DIALOG statement.
- By default, a new dialog has the Center Dialog checkbox enabled.

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

To insert a control into a dialog box using the click method

1. Click Control and then click the control you want to insert.
2. Position the mouse pointer in the dialog box.

   The mouse appears in the Control state (") when positioned in a dialog editor window.
3. Click in the dialog box where you want to place the control's top-left corner.

   The control is inserted with its default size settings.

Note

- To insert a control a multiple number of times, hold down CTRL while you make your selection. Press ESC to return the mouse back to the Selector state (").

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

To insert a control into a dialog box using the click & drag method

1. Click Control and then click the control you want to insert.
2. Position the mouse pointer in the dialog box.

   The mouse appears in the Control state (") when positioned in a dialog editor window.

3. Position the mouse pointer where you want one corner of the control to appear
4. Hold the mouse button down and drag up or down on a diagonal.
5. When the control is the size and shape you want, release the mouse button.

Note

- To insert a control a multiple number of times, hold down CTRL while you make your selection. Press ESC to return the mouse back to the Selector state (
").
- The control is inserted with a default label and identifier which you can change.

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

To undo operations

- Click Edit, Undo.

Note

The following actions cannot be reversed using undo:

- any change of view (activating windows, scrolling etc.)
- any file operation (open, saving, importing etc.)
- any object selection operations
- any operation with the Grid

rt: a_edit

To restores change reversed by the Undo command

- Click Edit, Redo.

rt: a_edit

To delete controls

1. Select the controls you want to delete.
2. Click Edit, Delete.

   The selected controls are not transferred from the dialog box to the clipboard.

Note

- Instead of using Edit, Delete, you can delete controls by clicking Edit, Cut which transfers controls from the dialog box to the clipboard as Corel SCRIPT statements.

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

To cut controls to the clipboard

1. Select the controls you want to cut.
2. Click Edit, Cut.

   The selected controls are transferred from the dialog box to the clipboard as Corel SCRIPT statements.

Note

- Instead of using Edit, Cut, you can delete controls by clicking Edit, Delete. This does not transfer controls from the dialog box to the clipboard.

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

To copy controls to the clipboard

1. Select the controls you want to copy.
2. Click Edit, Copy.
   The selected controls are copied from the dialog box to the clipboard as Corel SCRIPT statements.

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

To paste a copy of a control in a dialog box

1. Select the control(s) you want to copy.
2. Click Edit, Copy.

   The selected control(s) is copied from the dialog box to the clipboard as Corel SCRIPT statements.
3. Click in the dialog editor window you want to paste the control(s) into.
4. Click Edit, Paste.

Note

- A pasted control retains the original's label, identifier, size, and position attributes.
- If you paste a control into the same dialog editor window it was copied from, the pasted control is placed on top of the original control with the same name and identifier. You'll have to change the identifier in one of the controls because identifiers must be unique in a dialog definition.
- If you try to paste controls from a different sized dialog box into a position that doesn't exist in the second dialog box, the controls are placed in the closest valid position.
- If the controls are too big to fit into the second dialog box, the controls are resized to fit.
- If you paste Clipboard contents that contain the BEGIN DIALOG and END DIALOG statements, a new dialog box is opened in another dialog editor window.

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

To move a control in a dialog box using the mouse

1. Select the control(s) you want to move.
2. Press and hold the mouse button.
3. Drag the control(s) to a new location.
4. Release the mouse button.

Note

- As you are moving the controls, the Status Line shows the control coordinates of the last control you selected. The last control you select has a dotted line border.
- If Snap to Grid is enabled, the control moves along the dialog box grid.
- If Snap to Grid is enabled and more than one control is selected, the last control you selected moves along the dialog box grid.
- Pressing ESC while moving the control(s) resets the control(s) to its original position.

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

To move a control in a dialog box using the attributes box

1. Select the control(s) you want to move.
2. Click Edit, Attributes.
3. Type a number in the X number box for the distance from the inside of the dialog box's left border to the control's left border. The number is based in dialog units.
4. Type a number in the Y number box for the distance from the bottom of the dialog box's title bar to the control's top border. The number is based in dialog units.

Note

- If more than one control is selected, the X and Y number boxes are cleared because they cannot show a mixed-value. Though cleared, the number boxes accept entries. Entering values in the X and Y number boxes positions the top-left corners of the selected controls on the on the same X-Y coordinate in the dialog box.

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

To move a control from one dialog box to another

1. Select the control(s) you want to move.
2. Click Edit, Cut.
3. By clicking in it, activate the dialog editor window that you want to move the control(s) into.
4. Click Edit, Paste.

   The control(s) are placed in the same position as in the first dialog box.

Note

- If you try to move controls from a different sized dialog box into a position that doesn't exist in the second dialog box, the controls are placed in the closest valid position.
- If the controls are too big to fit into the second dialog box, the controls are resized to fit.

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

To resize a control in a dialog box using the mouse

1. Select the control(s) you want to resize.
2. Press and hold the mouse button on a sizing handle.

   The sizing handles on the corners change both the width and height. The other sizing handles change either the width or height.
3. Drag the handle until the control is the size you want.
4. Release the mouse button.

Note

- As you are resizing a control, the Status Line shows the control's new size and coordinates. If more than one control is selected, the Status Line shows the size of the last control you selected.
- If Snap to Grid is enabled, the control is resized along the dialog box grid.
- If Snap to Grid is enabled and more than one control is selected, the last control you selected is resized along the dialog box grid.
- Pressing ESC while resizing resets the control(s) to its original size.

{button ,AL(`a_edit;htde_control_size_label;a_select;;;',0,"Defaultoverview",)} Related Topics

To resize a control in a dialog box using the attributes box

1. Select the control(s) you want to resize.
2. Click Edit, Attributes.
3. Type a new value in the Width number box to change the width. The value is expressed in dialog units.
4. Type a new value in the Height number box to change the height. The value is expressed in dialog units.

Note

- Steps 3 and 4 are both optional. You can do one or both.
- If more than one control is selected, they are resized to the same width and height as specified in the Multiple Selection Attributes dialog box.

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

To edit a control's attributes using the attributes box

1. Select the control(s) you want to edit.
2. Click Edit, Attributes.
3. Type a new values into any of the following attribute boxes (if applicable):

- Width
- Height
- X (left border position)
- Y (top border position)
- Text
- Value
- Option Group
- Comment (a remark is added to the Corel SCRIPT statement for the control).

Note

- If more than one control is selected, the selected controls take on the attributes specified in the Multiple Selection Attributes dialog box.

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

To duplicate a control

1. Select the control(s) you want to duplicate.
2. Click Edit, Duplicate.

Note

- The duplicated control(s) is offset from the original by 3 dialog units, both down and to the right.
- The duplicated control(s) takes on the default identifier and the original control's label. If you copied the control, the original's label and identifier would also be copied.

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

To test a dialog box

- Click View, Test Dialog.

In test mode you can confirm the following dialog box features:

- tab order within the dialog box
- shortcut keys are operational
- drop-down boxes openings

Note

- Press ESC to exit test mode. Pressing any push button or the Close Dialog button (
-) also exits test mode.
- You cannot edit a dialog box in test mode.

The following controls are filled with place holders in test mode:

- list boxes
- drop-down list boxes
- combo boxes
- drop-down combo boxes

The place holders give you a better idea of what the dialog box will actually look like when it is run rather than does an empty control.

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

To select a control

1. Click the Control, Selector. By default, the mouse is in Selector mode and appears as ▪.
2. In the dialog document window, click anywhere on a control.

Note

- A selected control has a 8 sizing handles.

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

To select a dialog box

- Click anywhere in a dialog box until a thick border is displayed around it.

Note

- A thick border around a dialog box indicates it has been selected. Dialog boxes don't use sizing handles.

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

To select multiple controls

1. Click the Control, Selector. By default, the mouse is in Selector mode and appears as ▪.
2. In the dialog document window, while holding down the SHIFT key, click the controls you want to select.

Note

- Selecting more than one control lets you apply the same attributes to each of them.
- A selected control has a 8 sizing handles.
- You can also select multiple controls using a marquee select.
- The last control you select has a dotted line border.

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

To marquee select controls

1. Click the Control, Selector. By default, the mouse is in Selector mode and appears as ▪.
2. Hold down the mouse button and drag the marquee box until it completely encloses the controls you want selected.
3. Release the mouse button.

Note

- When you hold down ALT while you drag, any control that intersects with the marquee box is selected. If you do not release the mouse button before releasing ALT, only those controls enclosed by the marquee box will be selected.
- A selected control has a 8 sizing handles.
- The last control you select has a dotted line border.

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

To add or remove controls to a group of selected controls

1. Click the Control, Selector. By default, the mouse is in Selector mode and appears as ▪.
2. In the dialog document window, while holding down the SHIFT key, click the controls you want to select or de-select.

Note

- A selected control has a 8 sizing handles.
- The last control you select has a dotted line border.

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

To deselect all controls

- Click any open space outside the dialog box in the dialog editor window.

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

To select all controls in a dialog

- Click Edit, Select All.

Tip

- Selecting the dialog box selects all the controls in the dialog as well.
- Selecting all the controls in a dialog also selects the dialog box.
- The last control you select has a dotted line border.

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

To invert a selection of controls

1. Click Control, Selector. By default, the mouse is in Selector mode and appears as ▪.
2. Hold down SHIFT.
3. Hold down the mouse button and drag the marquee box until it completely encloses both the controls you want to select and those you want to de-select (invert select).
4. Release SHIFT.
5. Release the mouse button.

   The controls which were originally not selected are now selected and the controls which were selected are now not selected.

Note

▪ A selected control has a 8 sizing handles.

RT: a_select

To align controls along their left edge

1. Select the controls you want to align.
   The last control you select maintains its position; all other selected controls move to align with this control.
2. Click Arrange, Align Control, Left.

Note

- The last control you select has a dotted line border.

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

To align controls along their right edge

1. Select the controls you want to align.
   The last control you select maintains its position; all other selected controls move to align with this control.
2. Click Arrange, Align Control, Right.

Note

- The last control you select has a dotted line border.

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

To align controls along their top edge

1. Select the controls you want to align.
   The last control you select maintains its position; all other selected controls move to align with this control.
2. Click Arrange, Align Control, Top.

Note

- The last control you select has a dotted line border.

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

To align controls along their bottom edge

1. Select the controls you want to align.
   The last control you select maintains its position; all other selected controls move to align with this control.
2. Click Arrange, Align Control, Bottom.

Note

- The last control you select has a dotted line border.

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

To make controls the same width

1. Select the controls you want to resize.
   The last control you select maintains its width; all other selected controls resize to this control.
2. Click Arrange, Make Same Size, Width.

Note

- The last control you select has a dotted line border.

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

To make controls the same height

1. Select the controls you want to resize.
   The last control you select maintains its height; all other selected controls resize to this control.
2. Click Arrange, Make Same Size, Height.

Note

- The last control you select has a dotted line border.

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

To make controls the same width and height

1. Select the controls you want to resize.
   The last control you select maintains its size; all other selected controls resize to this control.
2. Click Arrange, Make Same Size, Both.

Note

- The last control you select has a dotted line border.

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

To size a control to fit its label

1. Select the control(s) you want to resize.
2. Click Arrange, Size to Content.

Note

- You can use this command on option buttons, check boxes, push buttons, and text.

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

To center controls vertically in a dialog box

1. Select the controls you want to center.
2. Click Arrange, Center in Dialog, Vertical.
   The selected controls are centered vertically based on the topmost and bottommost borders of the selected controls.

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

To center controls horizontally in a dialog box

1. Select the controls you want to center.
2. Click Arrange, Center in Dialog, Horizontal.
   The selected controls are centered horizontally based on the leftmost and rightmost borders of the selected controls.

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

To even the spacing between controls horizontally within a dialog box

1. Select the controls you want to arrange.
2. Click Arrange, Distribute, Horizontal.

   The selected controls are spaced evenly between the leftmost and rightmost borders of the selected controls.

Note

- More than two controls must be selected for this option to be enabled.

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

To even the spacing between controls vertically within a dialog box

1. Select the controls you want to arrange.
2. Click Arrange, Distribute, Vertical.
   The selected controls are spaced evenly between the topmost and bottommost borders of the selected controls.

Note

- More than two controls must be selected for this option to be enabled.

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

To turn on or turn off Snap to Grid

- Click Arrange, Snap to Grid.

Note

- Repeat the above procedure to turn off Snap to Grid
- When Snap to Grid is enabled, a control can only be moved along the dialog box grid. Snap to Grid can be enabled without showing the grid.
- Snap to Grid is set for all dialog editor windows.

{button ,AL(`a_edit;a_snap;a_insert;a_align;;',0,"Defaultoverview",)} Related Topics

To view or hide the grid

1. Click View, Grid Settings.
2. Enable Show Grid

   Show Grid is enabled when a check mark appears beside Show Grid.

Note

- Click Show Grid again to hide the grid. Show Grid is disabled when no check mark appears beside Show Grid.
- When Snap to Grid is enabled, a control can only be moved along the dialog box grid. Snap to Grid can be enabled without showing the grid.
- Show Grid is set for all dialog editor windows.

{button ,AL(`a_edit;a_snap;a_insert;a_align;;',0,"Defaultoverview",)} Related Topics

To set grid spacing

1. Click View, Grid Settings.
2. Type a new value in the Horizontal number box to set the horizontal spacing.
3. Type a new value in the Vertical number box to set the vertical spacing.

Note

- Grid measurements are expressed in dialog units.
- To show the grid, enable Show Grid in the same dialog box.
- Grid spacing settings are set for all dialog editor windows.

{button ,AL(`a_edit;a_snap;a_insert;a_align;;',0,"Defaultoverview",)} Related Topics

To resize the Script window

Within a dialog editor window, the Script window is attached below the dialog box window. By default the Script window is minimized when the Corel SCRIPT Dialog Editor is launched.

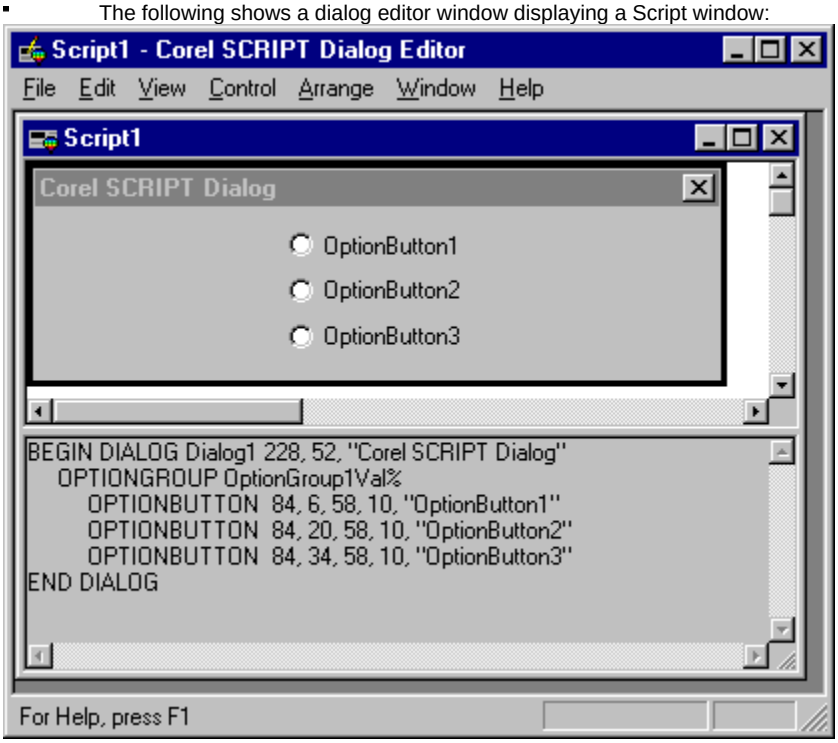
1. Click Window, Split.

   The mouse pointer changes to a two-headed arrow ( ).

2. Horizontally drag the top border of the Script window to the desired position.

Note

- The order of the statements in the Script window is determined by the order in which the controls were added to the dialog box.
- You cannot edit in the Script window but you can copy text to the clipboard..
- The following shows a dialog editor window displaying a Script window:



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

To view all dialog editor windows

- Click Window, Tile.

Note

- Minimized dialog editor windows are arranged at the bottom of the Corel SCRIPT Dialog Editor window.

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

To cascade dialog editor windows

- Click Window, Cascade.

Note

- Minimized script windows are arranged at the bottom of the Corel SCRIPT Dialog Editor window.

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

To arrange minimized dialog editor windows

- Click Window, Arrange Icons

Note

- Minimized dialog editor windows are arranged from the bottom-left corner of the Corel SCRIPT Dialog Editor to the bottom-right corner.

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

To view an open script window in the Corel SCRIPT Dialog Editor

- Click Window and click on the window you want to view.

The open windows are listed at the bottom of the Window menu.

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

To add a shortcut key to a control that has a label

1. Select a control that has a label (option buttons, push buttons, check boxes).
2. Click Edit, Attributes.
3. In the Label text box place an ampersand (&) in front of the letter you want to use as a shortcut key.

Note

- A shortcut key is an alternative to using TAB to move within a dialog and change focus. To use a shortcut, press ALT+***!*** where ***!*** is the defined shortcut key.
- Shortcut keys in a dialog box should be unique.

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

To add a shortcut key to a control that doesn't have a label

The text control that precedes a control without a label (for example, list boxes and text boxes) can be used to hold the shortcut for the control. The text control statement must immediately precede the control's statement in the dialog box definition. Before you insert a control without a label into a dialog box, you should insert a text control if you want to create an association between the unlabelled control and the text control. To add a shortcut key to a control that doesn't have a label:

1. Select the text control that immediately precedes the unlabelled control.

   Physical location in the dialog box is not important; it is the location of the Corel SCRIPT statement in the script that is important.
2. Click Edit, Attributes.
3. In the Label text box place an ampersand (&) in front of the letter you want to use as a shortcut key.

Note

- You can see if a text control precedes an unlabelled control by opening the Script window or copying the dialog box to the Corel SCRIPT Editor.
- A shortcut key is an alternative to using TAB to move within a dialog and change focus. To use a shortcut, press ALT+***!*** where ***!*** is the defined shortcut key.
- Shortcut keys in a dialog box should be unique.

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

To change dialog tab order

Tab order in a dialog box is based on the order in which the controls are listed in the Corel SCRIPT script. The underlying Corel SCRIPT statements for controls are stored in the order in which they were inserted into a dialog box.

- To change the order of the controls in the script file, transfer the dialog box to the Corel SCRIPT Editor and reorder the dialog box definition statements.

Note

- You view a dialog box's underlying Corel SCRIPT statements by opening the Script window.

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

To start Help from Corel SCRIPT Dialog Editor

- Click Help, Help Topics.

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

To open on-line Help to a selected control's syntax reference

1. Click Help, What's This
2. Click a control.

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

To view or hide status bar (Corel SCRIPT Dialog Editor)

- Click View, Status Bar.

A checkmark beside the Status Bar menu command indicates the status bar is displayed.

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

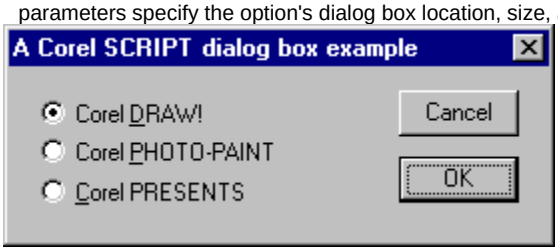
Corel SCRIPT and dialog boxes

Many times you need to get information from the user before your script performs a desired action. For simple information, you can use the Corel SCRIPT function INPUTBOX to get a string from the user returned to a running script. If you want to provide the user with options and more complex information, such as a list of choices, you can use a dialog box in your script.

Dialog boxes are created using the Corel SCRIPT language. The Corel SCRIPT language features a full set of programming statements to produce dialog boxes which incorporate sophisticated Windows options and features. For example, the Corel SCRIPT statements below create the following dialog box:

```
BEGIN DIALOG Dialog1 55, 10, 180, 53, "A Corel SCRIPT dialog box example"
    OKBUTTON 130, 27, 40, 14
    CANCELBUTTON 130, 7, 40, 14
    OPTIONGROUP ogroup%
        OPTIONBUTTON 10, 9, 90, 10, "Corel &DRAW!"
        OPTIONBUTTON 10, 21, 90, 10, "Corel &PHOTO-PAINT"
        OPTIONBUTTON 10, 33, 90, 10, "&Corel PRESENTS"
END DIALOG
```

The first and last statement initialize and end a dialog box's definition. Each indented statement specifies a dialog option, and the parameters specify the option's dialog box location, size, and other attributes



You have two options to create the Corel SCRIPT statements used to produce a dialog box. Your first option is to use the Corel SCRIPT Editor and type in the dialog statements. This can prove to be a time-consuming option because each statement's parameters are particular and it is difficult to visualize the dialog based on coordinate positions.

Your second option to create Corel SCRIPT statements used to produce a dialog box is to use the Corel SCRIPT Dialog Editor. With the Dialog Editor, you draw what you want your dialog box to look like. The dialog box, and the items within it, are graphical representations of Corel SCRIPT statements. The dialog box can be transferred to the Clipboard as Corel SCRIPT statements or saved as a Corel SCRIPT script file. With the Corel SCRIPT Dialog Editor you can quickly create or edit a dialog box in a few steps.

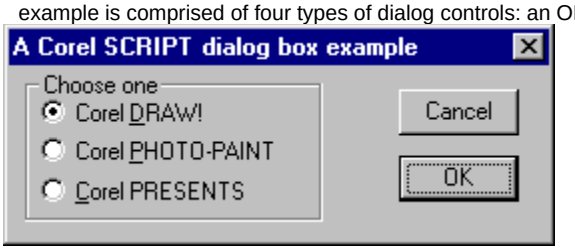
Note

The Corel SCRIPT Dialog Editor is a stand-alone program included with Corel applications. If you didn't install the Dialog Editor when you installed the Corel applications, you can run the Corel setup program again to install it. The Dialog Editor can be started from the Corel SCRIPT Editor or from the Windows desktop.

{button ,AL(`Corel_SCRIPT_dialog_controls;Dialog_controls_summary;Returning_dialog_settings_and_choices;Corel_SCRIPT_Dialog_Editor;Corel_SCRIPT_Editor;',0,"Defaultoverview",)} Related Topics

Corel SCRIPT dialog controls

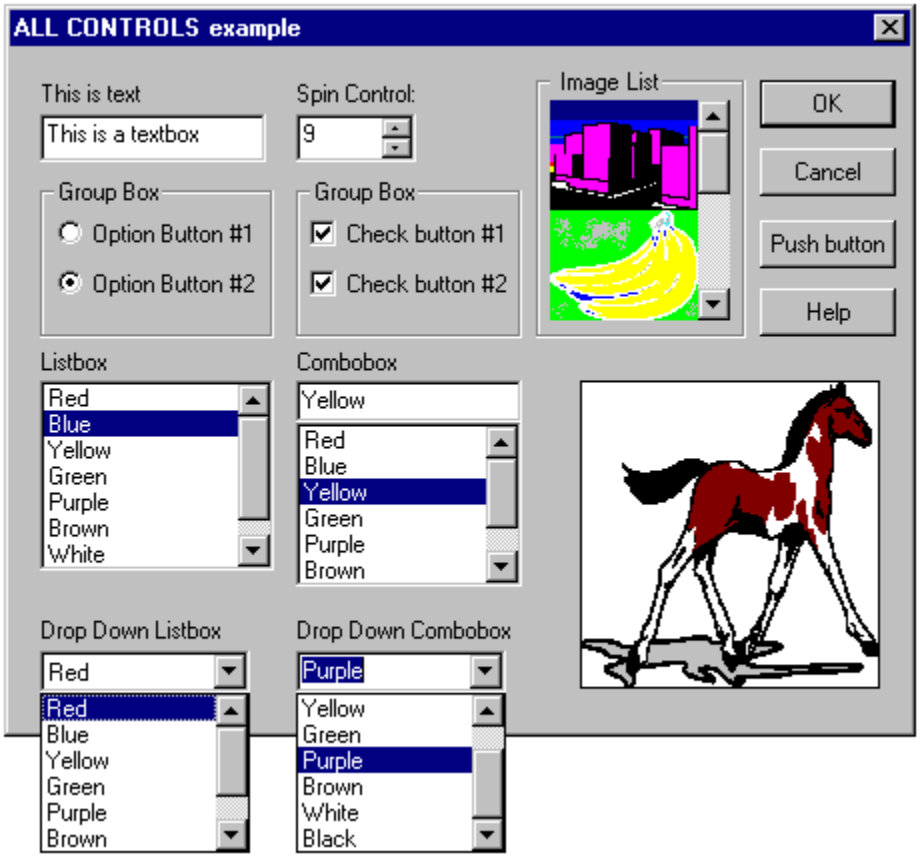
A Corel SCRIPT dialog box is comprised of a border that encloses graphic elements called dialog controls. By using the mouse and keyboard, you can interact with the controls by selecting options, entering text, and pressing dialog buttons. The following example is comprised of four types of dialog controls: an OK button, a Cancel button, three option buttons, and a group box.



Each control has its own set of underlying attributes. Each of the four types of controls above, and almost every other dialog control, have a height, width, horizontal location, vertical location, and text attribute. Additionally, most controls also have a value attribute. The value is a variable which that is assigned string and number values that reflect the state of the dialog box when it closes. These variables can then be used in a script to initiate actions dependent on their values.

In some cases controls don't return a value to a script because their only function is to organize the dialog box controls or to provide the user with information. In the above example, the group box is the only control that doesn't return a value to a running script but provides a logical grouping for the option buttons.

The following dialog box provides an example of every dialog control available in Corel SCRIPT. See Dialog controls summary for more information.



{button ,AL(`Dialog_controls_summary;Returning_dialog_settings_and_choices;Corel_SCRIPT_Dialog_Editor;Sizing_and_placing_dialogs_and_controls;Default_control_sizes_and_labels;',0,"Defaultoverview",)} Related Topics

Dialog controls summary

The following table lists all the dialog controls available in Corel SCRIPT. The last three entries note the other Corel SCRIPT statements that must be used to create and display a dialog box.

Control (explanation)	Syntax (syntax reference)	Return Value	Example (click pop-up)	Dialog Editor Control Button
Cancel button	CANCELBUTTON	2 and closes the dialog		
Check box	CHECKBOX	0 if disabled 1 if enabled 2 if mixed state		
Combo box	COMBOBOX	string value		
Drop-down combo box	DDCOMBOBOX	string value		
Drop-down list box	DDLISTBOX	integer value corresponding to an array		
Group box	GROUPBOX	doesn't return a value		
Help button	HELPBUTTON	Opens a help file at a specified topic (does not close dialog box)		
Image List Box	IMAGELISTBOX	string value		
Image	IMAGE	doesn't return a value		
List box	LISTBOX	integer value corresponding to an array		
OK button	OKBUTTON	1 and closes the dialog		
Option button	OPTIONBUTTON	value returns to OPTIONGROUP not the dialog		
Option group	OPTIONGROUP	integer value corresponding to the option button selected		
Push button	PUSHBUTTON	ID value (cannot equal 1 or 2) and closes the dialog		
Spin control	SPINCONTROL	numeric value		
Text box	TEXTBOX	string value		
Text	TEXT	doesn't return a value		
	BEGIN DIALOG	doesn't return a value but begins dialog box definition		
	END DIALOG	doesn't return a value but ends dialog box definition		
	DIALOG	doesn't return a value but displays dialog box		

{button ,AL(`Corel_SCRIPT_dialog_controls;Returning_dialog_settings_and_choices;Working_with_dialog_controls;Default_control_sizes_and_labels;Corel_SCRIPT_Dialog_Editor;',0,"Defaultoverview",)} Related Topics

Returning dialog settings and choices to a script

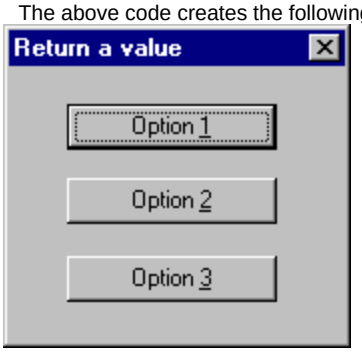
Dialog boxes are used in scripts so users can choose options and provide information back to the script. Once a dialog box closes, values are returned back to the script based on the user's interaction with the dialog box. User interaction includes selecting options, entering text, and pressing buttons. Each control that can accept an action stores a string or number value in its identifier variable. The identifier variables are then used in the script to initiate action dependent upon their values.

Example

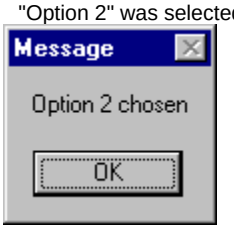
The following example shows how a dialog box can return a value back to a script.

```
' The dialog definition begins
BEGIN DIALOG Dialog1 19, 5, 112, 84, "Return a value"
    PUSHBUTTON 19, 11, 70, 14, "Option &1"
    PUSHBUTTON 19, 34, 70, 14, "Option &2"
    PUSHBUTTON 19, 58, 70, 14, "&Option &3"
END DIALOG
' the dialog definition ends
'
' the next line defines the dialog return variable called "returns"
x = DIALOG(Dialog1)
'
' If x is 3, then Option 1 was chosen
IF x = 3 THEN MESSAGE "Option 1 chosen"
' If x is 4, then Option 2 was chosen
IF x = 4 THEN MESSAGE "Option 2 chosen"
' If x is 5, then Option 3 was chosen
IF x = 5 THEN MESSAGE "Option 3 chosen"
```

The above code creates the following dialog box:



The above code also returns a message box dependent on the button pressed. The following message box is displayed if the "Option 2" was selected:



The variable "x" holds the dialog return value and is used in conditional statements to initiate further actions.

Note

- Each Corel SCRIPT dialog control that returns a value, does so in its own manner. For more information on how a dialog control returns a value, see its syntax reference and its respective example. See Dialog controls summary for a complete listing of Corel SCRIPT dialog controls.
- Corel SCRIPT dialog boxes are modal; that is, the running script cannot continue until the dialog box is closed.
- Neither the BEGIN DIALOG nor the END DIALOG statement on its own can display a dialog box; the DIALOG function displays a dialog box after it has been defined with the BEGIN DIALOG and END DIALOG statements.

{button ,AL(`Corel_SCRIPT_dialog_controls;Dialog_controls_summary;Transferring_and_saving_Corel_SCRIPT_state

ments;Viewing_a_dialog_boxs_Corel_SCRIPT_statements;Working_with_dialog_controls;',0,"Defaultoverview",)}
Related Topics

Corel SCRIPT Dialog Editor

The Corel SCRIPT Dialog Editor is a tool to quickly create and edit Corel SCRIPT dialog boxes. Working with the Dialog Editor is similar to using a drawing or painting application: dialog controls are graphic objects which can be inserted, moved, re-sized, and aligned in a dialog box.

Using the Dialog Editor takes the place of creating and editing Corel SCRIPT statements in a script file. It involves a visual approach to creating and editing Corel SCRIPT dialog statements, since a dialog box and the controls within it, represent Corel SCRIPT statements. From the Dialog Editor, you can transfer the dialog boxes you've created to the Clipboard, save the dialog boxes as Corel SCRIPT script files, and insert Corel SCRIPT statements into a script.

Using the Corel SCRIPT Dialog Editor

To create and edit dialog boxes, a dialog editor window must be open in the Dialog Editor. You can have more than one dialog editor window open, but each window can only hold one dialog box. Having more than one window open enables you to copy dialog boxes and controls between windows.

Each time you create new dialog, a dialog editor window opens containing a template of a dialog box. The template is a dialog box without any controls placed in it. From the dialog editor window you can re-size the dialog box, add and edit controls, and move the dialog box. When you move the dialog box within the dialog editor window, you are actually changing the dialog box's screen placement when it is run in a script. Its placement changes because a dialog editor window represents your computer screen, and moving the dialog box changes its horizontal and vertical position attribute. For example, if the dialog box is placed in the bottom-right corner of its dialog editor window it will appear on the bottom-right of a computer screen when it is run in a script file.

Existing script statements which create dialog boxes can be transferred to the clipboard and pasted into the Dialog Editor as a dialog box. Alternatively, you can open a Corel SCRIPT script file as a dialog box if it contains only dialog box definition statements.

Other features

You can use the Corel SCRIPT Dialog Editor to create, edit, and test dialog boxes. Dialog boxes, however, are just boxes without any controls in them. The Dialog Editor provides features for performing many activities associated with dialog box controls, including:

- inserting and deleting controls
- moving and resizing controls
- cutting, copying and pasting controls
- arranging and distributing controls
- setting control attributes and identifiers
- testing controls

Note

- The Corel SCRIPT Dialog Editor is a stand-alone program included with Corel applications. If you didn't install the Dialog Editor when you installed the Corel applications, you can run the Corel setup program again to install it. The Dialog Editor can be started from the Corel SCRIPT Editor or from the Windows desktop.

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

Working with the Dialog Editor

The easiest way to work in the Corel SCRIPT Dialog Editor is to use the mouse instead of the keyboard to create and edit dialog controls. The mouse pointer in the Dialog Editor has two states: the Selector state (default status) and the Control state.

In the Selector state the mouse pointer appears as ▪ and is used to select, re-size and drag-and-drop dialog controls. You can also select, move and resize dialog boxes when in the Selector state. By default, the mouse pointer is in the Selector state, and when not in the Selector state can be reset by clicking ▪ from the tool bar or clicking Control, Selector from the menu system.

In the Control state the mouse pointer appears as ▪ and is used to create dialog controls. The mouse pointer can only be set to a Control state by clicking any of the dialog controls in the Control ribbon bar or any control under the Control menu.

Note

The mouse pointer can only appear in a Control state when it is positioned over an active dialog box in a dialog editor window.

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

Inserting and deleting dialog controls

Inserting

To insert dialog controls into a dialog box, the mouse pointer must be in the Control state. To activate the Control state, select a control from the Control menu or from the Control ribbon bar. Once in the Control state, the selected control can be added to the dialog box by a clicking in the dialog box where you want the top-left corner of the selected control to appear. Each control has a default size setting. You can also use the click & drag method to insert controls into a dialog box.

After the control has been inserted, the mouse pointer reverts back to the Selector state. You can insert multiple controls of the same type by holding down the CTRL key as you select a control from the Control menu or from the Control ribbon bar. The mouse pointer remains in the Control state until you press ESC or select the Selector Tool from the menu or ribbon.

Along with a default size setting, most controls have default text and value attributes when inserted into a dialog box. The text labels and values update for each instance of the control inserted into a dialog box. For example, the first push button inserted takes the text **PushButton1**. The second push button takes on the text **PushButton2**, and so on.

Deleting

Controls are not permanent fixtures in a dialog box and can be deleted if not required.

{button ,AL(`csde;Dialog_controls_summary;Corel_SCRIPT_Dialog_Tips;Dialog_box_conventions;;',0,"Defaultoverview",)} Related Topics

Editing dialogs and controls

Dialogs and their controls can be moved, copied, deleted, resized, and their attributes edited through a variety of methods.

Moving and Resizing

By selecting a dialog box or control, you can move it or resize it using the mouse. Alternatively, opening a dialog box's or control's attribute box enables you to set position and size attributes.

Cut, Copy, Paste and Duplicate

The standard Cut, Copy and Paste Windows commands also function in the dialog editor with respect to dialogs and their controls. Additionally, the Duplicate command can help speed up your dialog box editing.

Label and identifier editing

You can change a dialog box's or control's label and identifier setting by opening its attributes dialog box and editing the settings.

Snap to Grid

If you're moving and resizing controls in a dialog box using the mouse, turning on Snap to Grid can help to accurately align and position controls. It also allows you to draw precisely-sized controls. The grid is not displayed during a script run.

Note

When a dialog box or a control is selected, the Dialog Editor displays its attributes in the status bar. The status bar also displays the mouse pointer position when positioned over a dialog box.

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

Aligning and distributing dialog controls

Dialog boxes that are arranged symmetrically, centered, and have controls aligned are easier to read and understand. The Corel SCRIPT Dialog Editor provides a complete set of tools to help you refine your dialog boxes. Using the tools you can:

- Align controls along an edge.
- Distribute controls evenly .
- Center controls in a dialog box.
- Make controls the same size.
- Size a control to its label.

Snap to Grid

If you're moving and resizing controls in a dialog box using the mouse, turning on Snap to Grid can help to accurately align and position controls. It also allows you to draw precisely-sized controls. The grid is not displayed during a script run.

Note

- When a dialog box or a control is selected, information about its attributes is displayed in the status bar. When the mouse pointer is over a dialog box, the status bar also displays information about the pointer's position.

{button ,AL(`csde;Dialog_controls_summary;Corel_SCRIPT_Dialog_Tips;Dialog_box_conventions;;',0,"Defaultoverview",)} Related Topics

Using dialog units to size and position dialogs and controls

Sizing dialogs and controls

Every dialog box and dialog box control has a width and height attribute, expressed in dialog units. For width, a dialog unit is 1/4 the average width of the Corel system font. For height, a dialog unit is 1/8 the average height of the Corel system font. In other words, a dialog unit for width and height are practically equal because, on average, the height of the Corel system font is twice its width (1/8 x 2=1/4). So if you create a dialog box that is 500 units (width) by 500 units (height), you should end up with a dialog box that is a square or very close to a square.

Positioning dialogs and controls

Dialog controls also have attributes which hold position measurements in a dialog box. A control's vertical position is measured in width dialog units from the inside of the dialog box's left border to the left side of the control. A control's horizontal position is measured in height dialog units from the bottom of the dialog box's title bar to the top of the control.

A dialog box's position during a script run is also set with attributes which hold dialog unit measurements. The position of the left border with respect to the left side of the monitor's display area is measured in width dialog units, and the position of the top border with respect to the top of the monitor's display area is measured in height dialog units.

A dialog box can also be set to be centered on the screen by either omitting the position attribute parameters in the BEGIN DIALOG statement or by enabling the Center Dialog check box in the attributes dialog box in the Dialog Editor.

When you create a new dialog box in the Dialog Editor, by default the dialog is placed in the top-left corner of the dialog editor window with the Center Dialog attribute enabled. If the Center Dialog attribute was disabled, the dialog box would be displayed in the monitor's top-left corner when run in a script, because the dialog editor window is a representation of your monitor. When you move the dialog box within the dialog editor window, you are actually changing the dialog box's screen placement when it is run in a script. If you move the dialog box to the bottom-right corner of its dialog editor window, it will appear on the bottom-right of a monitor when run in a script file.

Note

- The Corel system font cannot be changed.
- In some cases, changing the screen resolution will change a dialog box's appearance. If your dialog boxes will be used on a variety of screens at different resolution settings, you should test the dialogs at each setting to ensure they will be properly displayed.

{button ,AL(`csde;Dialog_controls_summary;Corel_SCRIPT_Dialog_Tips;Dialog_box_conventions;;',0,"Defaultoverview",)} Related Topics

Default control sizes and labels

The following table list's each control's default width and height settings in dialog units:

Control	Width	Height	Label
Text	50	8	TextN
Text box	50	13	N/A
OK button	40	14	OK
Cancel button	40	14	Cancel
Push button	46	14	PushButtonN
Help button	40	14	Help
Option button	58	10	OptionButtonN
Check box	50	10	CheckBoxN
List box	50	42	N/A
Drop-down list box	50	42	N/A
Combo box	50	42	N/A
Drop-down combo box	50	42	N/A
Group box	50	40	N/A
Spin control	50	12	N/A
Image List Box	50	42	N/A
Image	40	40	N/A

Note

- N is the Nth occurrence of the control in the dialog box.

{button ,AL(`csde;Dialog_controls_summary;Dialog_box_conventions;Corel_SCRIPT_dialog_controls;;',0,"Defaultoverview",)} Related Topics

Viewing a dialog box's Corel SCRIPT statements

Dialog boxes displayed in the Corel SCRIPT dialog editor are graphical representations of Corel SCRIPT statements. If you wanted to see the statements underlying the dialog box you could transfer a dialog to the Clipboard, then insert the Clipboard contents into a script file as Corel SCRIPT statements, or save the dialog as a script file.

An easier option exists to view the Corel SCRIPT statements underlying a dialog box in a non-editing mode. Each dialog editor window in the Dialog Editor has a sub-window which displays the statements that correspond to the dialog and controls in the dialog editor window. The sub-window is called the Script window and its contents can only be viewed; it has no editing function.

Though it has no editing functions, the Script window can provide you with a significant amount of information about a dialog box. If you are an inexperienced dialog user, you can view the results of your dialog box editing operations in terms of the Corel SCRIPT statements. If you are a more experienced user, the Script window can help you determine the grouping of your option buttons . You can also view the tabbing order of the dialog box, and text control associations with non-labeled controls. See TEXT for more information about text control associations.

The following shows a dialog editor window displaying a Script window:

Note

You can copy dialog box definition statements from the Script window to the clipboard. However, you cannot perform any editing function in the Script window.

The order in which controls are entered into a dialog box determines the order of the dialog box script statements.

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

Changing focus in dialog boxes

Within a dialog box, only one control can have focus. Focus means that the control is active and can accept user input such as text or option selections. A dialog box's initial focus goes to the first control defined after the BEGIN DIALOG statement in a dialog box definition.

To move around in a dialog box from control-to-control, you can use the TAB key. As you move around, the dialog's focus changes. A dialog convention is to have the focus move from left-to-right and top-to-bottom as you TAB through a dialog. The TAB order is based on the order in which the controls are defined in the dialog box definition statements. You should try to place your control statements between the BEGIN DIALOG and END DIALOG statements in the order in which you want to TAB through a dialog.

Shortcut keys

An alternative to using TAB to move around a dialog and change focus is to use shortcut keys. By placing an ampersand (&) before a character in a dialog control's label you can create a shortcut key. For example, in the following dialog box, pressing ALT+D changes the dialog focus to the first push button.

Dialog controls that don't have labels, such as text boxes or list boxes, can also take on shortcut keys if you create an association with a text control. To associate a text control with a dialog control, the text control statement in the Corel SCRIPT script must precede the unlabelled control statement. Association is not based on a control's dialog location.

The following example shows two text boxes with defined shortcut keys:


```
BEGIN DIALOG Dialog1 55, 10, 97, 89, "Shortcut key example"
    TEXT 5, 4, 80, 8, "&Enter text here:"
    TEXTBOX 5, 13, 80, 12, text1$
    TEXT 5, 31, 80, 8, "&Here too:"
    TEXTBOX 5, 40, 80, 12, text2$
    OKBUTTON 5, 64, 80, 14
END DIALOG
```

Note

The initial focus of a dialog box should be a text box control, so that a user can immediately start typing once the dialog opens.

Pressing ENTER is a shortcut for the OK button, and pressing ESC or the Close Dialog button ( ) is a shortcut for the CANCEL button.

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

Testing dialogs

In the Dialog Editor, you can test the dialog boxes you create to confirm they meet your requirements and function properly. It is easier to test a dialog within the Dialog Editor than by making it part of a script and running the script.

To test a dialog box, set the dialog editor window to test mode. In test mode you can confirm the following dialog box features:

- tab order within the dialog box
- shortcut keys are operational
- drop-down box openings

Note

- You cannot edit a dialog box in test mode.
- The following controls are filled with place holders in test mode:
  - list boxes
  - drop-down list boxes
  - combo boxes
  - drop-down combo boxes
  - static image boxes are always filled with a placeholder

  The place holders give you a better idea than an empty control of what the dialog box will actually look like when it is run.

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

Transferring and saving Corel SCRIPT dialog definitions

Working with the Dialog Editor involves a visual approach to creating and editing Corel SCRIPT dialog statements, since a dialog box and the controls within it represent Corel SCRIPT statements. Because dialogs and controls in the Dialog Editor represent Corel SCRIPT statements, they can be transferred to the Corel SCRIPT Editor and back.

Transferring statements from the Corel SCRIPT Editor to the Dialog Editor

It is often easier to edit a dialog box in the Dialog Editor instead of the Corel SCRIPT Editor. You can transfer dialog statements into the Dialog Editor from the Clipboard via the Editor. You cannot transfer statements that are not used in a dialog box definition. You can also transfer a dialog box definition by selecting it and opening the Corel SCRIPT Dialog Editor.

Transferring statements from the Dialog Editor to the Corel SCRIPT Editor

You can transfer dialog box and control statements from a dialog box to the Clipboard, then insert them into the Corel SCRIPT Editor.

Saving and opening dialog script statements

The Corel SCRIPT Dialog Editor saves a dialog box as a Corel SCRIPT script file containing Corel SCRIPT statements. The statements form a dialog box definition that can be inserted into another script file or used as the basis for building a script file.

You should save dialog box scripts in your application's Script folder. For example, C:\COREL60\SCRIPTS\DRAW for CorelDRAW scripts.

Opening scripts in the Dialog Editor

The Corel SCRIPT Dialog Editor can only open previously saved scripts that only contain a Corel SCRIPT dialog box definition.

{button ,AL(`csde;Corel_SCRIPT_Editor;Returning_dialog_settings_and_choices;;;',0,"Defaultoverview",)} Related Topics

Corel SCRIPT Dialog Tips

- In some cases, changing the screen resolution will change a dialog box's appearance. If your dialog boxes will be used on a variety of screens at different resolution settings, you should test the dialogs at each setting to ensure they will be properly displayed.
- Consider using the INPUTBOX, GETFILEBOX, MESSAGEBOX, or MESSAGE instead of a user-defined dialog box.
- Neither the BEGIN DIALOG nor the END DIALOG statement on its own can display a dialog box; the DIALOG function displays a dialog box after it has been defined with the BEGIN DIALOG and END DIALOG statements.
- You can insert multiple controls of the same type by holding down the CTRL key as you select a control from the Control menu or from the Control ribbon bar.
- The dialog Grid and the Snap to Grid options can help you align and distribute your dialog controls.
- List boxes and drop-down list boxes return integers that correspond to a selection from the array displayed in the control. The first array item returns 0, the second returns 1 and so on.
- Pressing the Close Dialog button (
") is the same as pressing the Cancel button; both return 2 to the dialog box.
- Use push buttons to open other dialog boxes. For example, within a script consider using a CASE statement to open another dialog box after a user has pressed a push button.
- Pressing ENTER is a shortcut for clicking the OK button, and pressing ESC or the Close Dialog button (
") is a shortcut for clicking the CANCEL button.
- You can select more than one dialog control by holding down the SHIFT key while selecting, or by clicking the mouse and dragging across the dialog box in the dialog editor window.
- Corel SCRIPT dialog boxes are modal; that is, the running script cannot continue until the dialog box is closed.

{button ,AL(`Dialog_box_conventions;Corel_SCRIPT_and_dialog_boxes;Corel_SCRIPT_Dialog_Editor;;;',0,"Defaultover view",)} Related Topics

Dialog box conventions

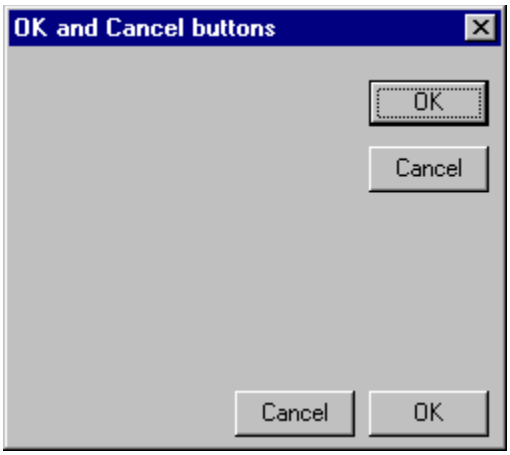
To make dialog boxes easier to understand at a quick glance, consider the following conventions when creating them:

- Dialog boxes are read left-to-right and top-to-bottom. The dialog controls should be arranged in a way that allows a user to easily read the dialog controls. The tab order should also follow the left-to-right and top-to-bottom convention.
- The initial focus of a dialog box should be a text box control so that a user can immediately start typing once the dialog opens.
- Use sentence formatting for dialog text titles; that is, the initial character of the sentence should be in uppercase. The sentence should end with a colon, not a period. For example, **Choose a file:**
- Push button text should only have the initial character of each word in uppercase characters. For example, **Close File** not **CLOSE FILE**. If a push button option opens another dialog, the push button text should be followed by an ellipsis (...). For example, **Printer Options...** .
- Group and position option buttons and check boxes vertically. Both option buttons and check boxes should be grouped in a logical manner. Avoid grouping option buttons and check boxes together.
- Consider using a list box of some type when you are providing more than 6 choices.
- Provide a shortcut key for every dialog control. For controls that don't use labels, such as text boxes and list boxes, create a shortcut key association with a text control.
- Static text should be aligned to the control it is identifying.
- Each dialog box should include both an OK and a Cancel button. The two buttons should be aligned horizontally in the bottom-right of the dialog box, or aligned vertically in the top-right of a dialog box.

Sizing, aligning and distributing dialog controls

- Dialog controls of the same type should be the same size, especially in the case of push buttons, check boxes, option buttons, and list boxes.
- Use a margin of 4 dialog units from the inside of the dialog box borders.
- Separate groups of controls by 6 units both horizontally and vertically. Controls within a group can be separated by 2 units, except for option buttons and check boxes which do not need any separation.
- Use the alignment and distribution tools to create a symmetrical look for dialog controls.

{button ,AL(`Corel_SCRIPT_Dialog_Tips;Corel_SCRIPT_and_dialog_boxes;Corel_SCRIPT_Dialog_Editor;Sizing_and_placing_dialogs_and_controls;Default_control_sizes_and_labels;',0,"Defaultoverview",)} Related Topics


OK and Cancel buttons
OK
Cancel
Cancel
OK

Dialog box definition statements

A dialog box definition consists of the BEGIN DIALOG and END DIALOG statements with control statements in between. The following list contains valid dialog box definition statements:

- BEGIN DIALOG
- CANCELBUTTON
- CHECKBOX
- COMBOBOX
- DDCOMBOBOX
- DDLISTBOX
- END DIALOG
- GROUPBOX
- HELPBUTTON
- IMAGE
- IMAGELISTBOX
- LISTBOX
- OKBUTTON
- OPTIONBUTTON
- OPTIONGROUP
- PUSHBUTTON
- SPINCONTROL
- TEXT
- TEXTBOX

No related topics were found.

No procedure topics were found.

Corel SCRIPT Dialog Editor menu shortcut keys

Press	To
File Menu	
CTRL+N	New
CTRL+O	Open
CTRL+V	Paste Dialog
CTRL+S	Save
Edit Menu	
CTRL+X	Cut
CTRL+C	Copy
CTRL+V	Paste
DEL	Delete
CTRL+D	Duplicate
CTRL+A	Select All
ALT+ENTER	Attributes
View Menu	
CTRL+T	Test Dialog
CTRL+G	Grid Settings
Arrange Menu	
F7	Size To Content
CTRL+Y	Snap To Grid
Help Menu	
F1	Help Topics

Syntax

Text control

Text controls in dialog boxes are used as labels and to provide user instructions. As labels, text controls cannot be changed by a user and do not return any value back to a running script.

Use sentence formatting for dialog text titles; that is, the initial character of the sentence in uppercase. The sentence should end with a colon, not a period. For example, **Choose a file:** You should provide text labels for any dialog control that doesn't have a label component.

Controls that use labels such as push buttons and option buttons can have shortcut keys assigned to them by placing an ampersand (&) before a label character. When the script displays the dialog box, you can use a keyboard shortcut to select a control by pressing ALT and the underlined shortcut key (the character that follows the ampersand). You can also use the text control to provide shortcut keys for controls that don't use labels.

A text control must be associated with an unlabelled dialog control (available controls noted below) to provide a keyboard shortcut. To associate a text control with a dialog control, the text control statement in the Corel SCRIPT script must precede the unlabelled control statement. Association is not based on a control's dialog location.

The following controls can be associated with a text control:

- text box
- list box
- drop-down list box
- combo box
- drop-down combo box
- spin control
- image list box

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

Syntax

Text box control

The text box control receives user-inputted text which is returned to the script as a string. If you want a number returned to a script from a text box, you'll need to use the VAL function to convert the text because a text box can only return strings. You can input up to 255 characters in a text box and use standard window shortcut keys such as **CTRL+X** to cut text, **CTRL+C** to copy text, and **CTRL+V** to paste text. Arrow keys can also be used in a text box to move the cursor.

Because a text box doesn't have a label component, a text control should be used to identify the text box control. The text control label can be used to provide a shortcut to the text box.

If the text box is a single line, you can scroll through the text horizontally using the arrow keys. If the text box is multi-line, you can scroll through the text horizontally and vertically using the arrow keys.

Pressing CTRL+ENTER in a text box inserts a line return character. Pressing ENTER or SHIFT+ENTER in the text box control is the same as pressing a push button, and closes the dialog box .

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

Syntax

OK button control

Pressing an OK button, a Cancel button, or a Push button closes a dialog box. Pressing an OK button closes a dialog box, assigns value settings to specified variables, and assigns 1 as the dialog box return value.

The OK button can't be assigned a shortcut key but pressing ENTER is the same as pressing the OK button (unless a push button or the Cancel button has focus). Only one OK button in a dialog box is suggested.

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

Syntax

Cancel button control

Pressing an OK button, a Cancel button, or a Push button closes a dialog box. Pressing a Cancel button closes a dialog box and discards the settings within it, and assigns 2 as the dialog box return value

The Cancel button can't be assigned a shortcut key but pressing the Close Dialog button (") is the same as pressing the Cancel button. Only one Cancel button in a dialog box is suggested.

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

Syntax

Push button control

Pressing an OK button, a Cancel button, or a Push button closes a dialog box. Push buttons are often used in cases in which another dialog box opens when a push button is pressed.

You can give a push button any string label to display. If the button is pressed, its assigned identifier number is returned by the DIALOG function. The Dialog Editor assigns an identifier number to each push button you create, beginning with 3 (1 and 2 are reserved for the OK and Cancel buttons). A push button's identifier number does not depend on its physical location within a dialog box but to the order in which it was inserted into the dialog.

Pushbuttons can be assigned keyboard shortcuts using the ampersand (&) character.

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

Option button control

Option buttons are used to present two or more mutually exclusive choices. Only one option button in a group can be selected and each selection de-selects a previously selected button. One button from a group is always selected.

Option buttons should be used to choose a property, or to set a value or an option. Consider using some type of list box when you have more than 6 choices. To make a dialog box easier to read, put a group box around option buttons and a label indicating the purpose of the option buttons. Each option button has a label component and a shortcut key can be assigned to it.

In a script, option buttons are grouped using the OPTIONGROUP statement. An option button group exists only when the option button statements are grouped together without intervening statements (remarks excluded). The first option button statement must be preceded by an OPTIONGROUP statement.

The Dialog Editor groups option buttons as you create them. As you insert option buttons, they will automatically be included in the same group until you enter another type of control. Once you enter another type of control and then insert more option buttons, the options buttons will form another group with an OPTIONGROUP statement preceding the first option button.

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

Option group control

An option group is not displayed in a dialog box. It is a Corel SCRIPT statement that marks the beginning of a series of option buttons. An option button group exists only when the option button statements are grouped together without intervening statements (remarks excluded) and the first option button statement is preceded by an OPTIONGROUP statement.

The Dialog Editor groups option buttons as you create them. As you insert option buttons, they will automatically be included in the same group until you enter another type of control. Once you enter another type of control and then insert more option buttons, the options buttons will form another goup with an OPTIONGROUP statement preceding the first option button.

You can select only one option button in a group. The option group identifier returns an integer (0 through n) that corresponds to the option button selected, with the first option button being equal to 0. The second option button is equal to 1and so on. The order of the option buttons is determined, not by their placement within the dialog box, but by their listing in the script.

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

Check box control

Unlike option buttons, check boxes offer non-exclusive choices for users. Check boxes have three states:

- disabled status (empty)
- enabled status (filled with a check mark)
- mixed status (gray-filled)

Users can cycle through the check box states by clicking the check box itself, or by clicking its label. A shortcut key can be associated with the check box. You can also set the default state of a check box. To make a dialog box easier to read, put a group box around related check boxes and a label on the group box to give the user more information.

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

List box control

A list box provides a user with a single choice from a restricted set of items. A list box should be used instead of option buttons when there are more than six option items.

Each item in a list box comes from a previously defined one-dimension array. The items should be listed in a logical order such as alphabetical, ascending sort for number values, or some other logical sort that is appropriate for the items in the list.

If the items can't vertically fit into the list box, the list box automatically takes on vertical scroll bar. The scroll bar is placed on the right-side of the list box to the immediate left of the right border. Having the scroll bar placed within the list box cuts into the item display area so you should ensure that each list box entry can at least be recognized.

Because a list box doesn't have a label component, a text control should be used to identify the list box . The text control label can also be used to provide a shortcut to the list box. A default selection can be provided in a list box and an integer associated with a selection is always returned to the script.

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

Drop-down list box control

A drop-down list box is like a list box except that you can only display the list box items on demand. That is, to display the items and to make a choice, you must open the drop-down list box. Once a selection is made the drop-down portion of the control closes.

The advantage of using a drop-down list box over a list box is that the control uses less space. In some cases this might make the dialog easier to read. The drawback is that it adds another level of user interaction to a dialog box.

Like the list box, the drop-down list box provides a user with single choice from a restricted set of items. Each item in a drop-down list box comes from a previously defined one-dimension array. The items should be listed in a logical order such as alphabetical, ascending sort for number values, or some other logical sort that is appropriate for the items in the list.

If the items can't vertically fit into the list box, the drop-down list box automatically takes on vertical scroll bar. The scroll bar is placed on the right-side of the list box to the immediate left of the right border. Having the scroll bar placed within the list box cuts into the item display area so you should ensure that each list box entry can at least be recognized.

Because a drop-down list box doesn't have a label component, a text control should be used to identify the drop-down list box . The text control label can also be used to provide a shortcut to the drop-down list box. A default selection can be provided in a drop-down list box and an integer associated with a selection is always returned to the script.

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

Combo box control

As its name indicates, the combo box is a combination of boxes; in this case a combination of a text box and a list box. From a combo box you can make a selection from a restricted set of items or enter your own selection in the text window. The advantage of a combo box over a list box is that you don't restrict a user to predefined items. A default selection can be provided in a combo box and a string associated with a selection is returned to the script.

If you type text in the text box, the list box scrolls to the closest matching item in the list box portion of the combo box. Choosing a list item from the list box replaces the text box contents with the list box selection.

Because a combo box doesn't have a label component, a text control should be used to identify the combo box . The text control label can also be used to provide a shortcut to the combo box.

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

Drop-down combo box control

A drop-down combo box is a combination drop-down list box and text box.

The advantage of using a drop-down combo box over a regular combo box is the control uses less space. In some cases this might make the dialog easier to read. The drawback is that it adds another level of user interaction to a dialog box.

If you type text in the text box, the list box scrolls to the closest matching item in the list box portion of the drop-down combo box. Choosing a list item from the list box replaces the text box contents with the list box selection.

Because a drop-down combo box doesn't have a label component, a text control should be used to identify the dialog control. The text control label can also be used to provide a shortcut to the drop-down combo box. A default selection can be provided in a drop-down combo box.

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

Group box control

The group box is a dialog control that doesn't return anything to a script but is used to help arrange dialog controls to make the dialog box easier to understand. The group box is most useful in physically grouping related check boxes and option buttons. The label component of the group box gives a user more information about a dialog.

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

Spin control

The spin control is used to change values in numeric entry text boxes by using the mouse or keyboard. The top arrow increases the value displayed, the bottom arrow decreases it. You can either click the arrow to change the value by a single increment or hold the mouse button down on an arrow to cause it to change continuously. You also have the option of typing directly into the text window portion.

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

Image list box control

The image list box is a dialog control used to preview and select Windows graphic files. The image list box can display Windows bitmaps (.BMP and .RLE files). You select an image by clicking it.

Like the list box, the image list box provides a user with a single choice from a restricted set of items. Each item in an image list box comes from a previously defined one-dimension array. Images displayed in an image list box are resized to fit horizontally.

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

Help button control

Use the help button control to open a Windows help file to a specified help topic. The help button control is an advanced control that should only be used by Windows programmers and Windows help file authors. To use the Help button control, you must know the context number ID values used with help topics.

The help button is not a push button and does not close a dialog box.

Note

For more information about creating and compiling Windows help files and context-sensitive help, consult the Windows SDK or the Windows Help Author's Guide.

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

Image control

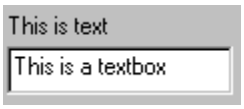
The static image control displays a specified Windows graphic in a Corel SCRIPT dialog box. The graphic is automatically sized to fit the height and width you've specified.

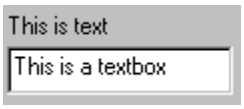
As a dialog control, an image control doesn't return a value to a script. The image control should be used to provide visual information to a user.

Note

- You can insert Windows bitmaps (.BMP and .RLE files) into an image control.

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

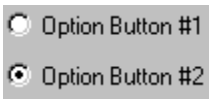

This is text
This is a textbox

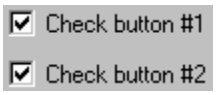

This is text
This is a textbox


OK

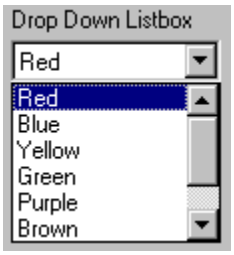

Cancel

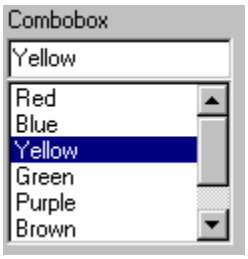

Push button


Option Button #1
Option Button #2

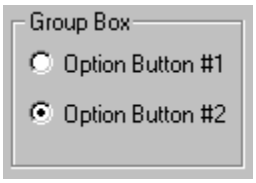

Check button #1
Check button #2

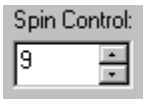

Listbox
Red
Blue
Yellow
Green
Purple
Brown
White


Drop Down Listbox
Red
Red
Blue
Yellow
Green
Purple
Brown


Combobox
Yellow
Red
Blue
Yellow
Green
Purple
Brown


Drop Down Combobox
Purple
Yellow
Green
Purple
Brown
White
Black


Group Box
Option Button #1
Option Button #2


Spin Control:
9


Level:

Under
Construction


Image List


Help

Save these for future use

Horizontal scroll bar control

Scroll bars should be used in cases where a window's contents cannot fit into its display area. A scroll bar enables you to pan a window so you can view different portions of it's contents. To pan a window, the scroll bar's scroll box should be moved. The scroll box is proportional, meaning that it varies in size based on how much of the window's contents can be viewed in the window.

Use the horizontal scroll bar in cases where a window's contents overflow to the left or right of a window.

Vertical scroll bar control

Scroll bars should be used in cases where a window's contents cannot fit into its display area. A scroll bar enables you to pan a window so you can view different portions of it's contents. To pan a window, the scroll bar's scroll box should be moved. The scroll box is proportional, meaning that it varies in size based on how much of the window's contents can be viewed in the window.

Use the vertical scroll bar in cases where a window's contents overflow to the top or bottom of a window.

Slider control

The slider is used for setting and adjusting continuous numeric values. Some examples of appropriate uses for the slider include using it to adjust size, volume, or color intensity.

The slider indicator is used to adjust the control's return value. You can change the slide indicator position by dragging it horizontally. You can also move the slider indicator by clicking along the bar or using the arrow key on the keyboard when the slider control has focus.

Because a slider doesn't have a label component, a text control should be used to identify the slider. The text control label can be used to provide a shortcut to the slider.

Progress indicator control

For operations that can take a significant amount of time, you may want to use the progress indicator. The progress indicator visually displays an operation's percentage of completion.

As a control, the progress indicator doesn't return a value to a script and is used to provide information to a user. Because a slider doesn't have a label component, a text control should be used to identify it.

Using date and time

Corel SCRIPT provides commands and functions to manipulate dates and time. Dates are one of Corel SCRIPT's variable types. They are explicitly declared using the **DATE** keyword and do not use a suffix. Date variables hold date and time values that range from 1 to 2958465 (as a serial number) or 31/12/1899 00:00:00.0000 to 31/12/9999 23:59:59.9999 (as a date value). You can use dates outside this range but they are not supported by Corel SCRIPT and may lead to errors.

Serial numbers represent both date and time values. Numbers to the left of the decimal represent days, and numbers to the right of the decimal represent time. If you omit decimal numbers, time is set to 12:00:00 A.M.

The following table shows examples of serial numbers using whole and decimal numbers to represent dates and time:

Serial number	Date and time represented
1	December 31, 1899, 12:00:00 A.M.
2	January 1, 1900, 12:00:00 A.M.
2.25	January 1, 1900, 6:00:00 A.M.
16229.2292	June 6, 1944, 5:30:00 A.M.
23337.4375	November 22, 1963, 10:30 A.M.
25882.6701	November 10, 1970, 4:04:57 P.M.
35064.9999	December 31, 1995 11:59:59 P.M.
35065	January 1, 1996, 12:00:00 A.M.
35065.5	January 1, 1996, 12:00:00 P.M.

The following table shows the decimal value for hours of the day:

Decimal serial number	Hour of the day represented
.0	12 A.M.
.125	3 A.M.
.25	6 A.M.
.375	9 A.M.
.5	12 P.M.
.625	3 P.M.
.75	6 P.M.
0.875	9 P.M.

Each hour is approximately equal to .041666

Note

- Date and time formatting is set in your Windows Regional Setting. The Corel SCRIPT date format style is based on the Windows Short date style, and the time style is based on the Windows time style. To change your Windows settings, click Start, Settings, Control Panel. Double-click Regional Settings and choose the folders you want to change.

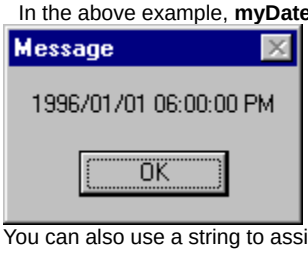
{button ,AL(`cs_date_time;;;;;',0,"Defaultoverview",)} **Related Topics**

Assigning values to date variables

You have two options to assign a date and a time to a date variable. One option is to use the serial number. For example:

```
DIM myDateVar AS DATE
myDateVar = 35065.75
MESSAGE myDateVar
```

In the above example, **myDateVar** is set to January 1, 1996 6:00 P.M. as shown in the following message box:



You can also use a string to assign a date and a time to a date variable. The following examples all set **myDateVar** to January 1, 1996 6:00 P.M.:

```
myDateVar = "01/01/96 18:00"
myDateVar = "January 1 1996 6:00 PM"
myDateVar = "1 January 96 6:00 PM"
myDateVar = "01-01-96 18:00"
```

The Corel SCRIPT date variable uses pre-defined logic to determine how to convert strings into dates. In ambiguous cases, for example, when a portion of string could be a day, month, or year, Corel SCRIPT refers to the system's date settings to determine how to convert the string.

Note

- Date and time formatting is set in your Windows Regional Setting. The Corel SCRIPT date format style is based on the Windows Short date style, and the time style is based on the Windows time style. To change your Windows settings, click Start, Settings, Control Panel. Double-click Regional Settings and choose the folders you want to change.

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

CURRDATE

The CURRDATE statement is used to assign the current system date and time to a variable, or to change the current system date.

Syntax to change the system date and time
CURRDATE = theDate

Syntax to assign the system date and time to a variable
theDate = CURRDATE

Syntax	Definition
theDate	The name of the date variable.

Note

- Date and time formatting is set in your Windows Regional Setting. The Corel SCRIPT date style is based on the Windows Short date style and the time style is based the Windows time style. To change your settings, click Start, Settings, Control Panel. Double-click Regional Settings and choose the folders you want to change.
- You can't set your operating system date beyond December 31, 2099.

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

Examples for CURRDATE

To display the system date

The following example displays the system date in a message box.

```
MESSAGE CURRDATE
```

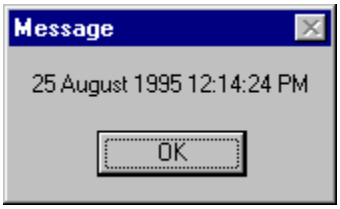
To change the system date and time:

The following example sets the system date to August 25, 1995 and the time to 12:00 A.M. A message box is used to display the date value of **dateOnly**.

```
DIM dateOnly AS DATE
dateOnly = 34936
CURRDATE = dateOnly
MESSAGE dateOnly
```

The following example sets the system date to August 25, 1995 and the sytem time to 12:05.46 P.M. A message box is used to display the date value of **dateAndTime.**

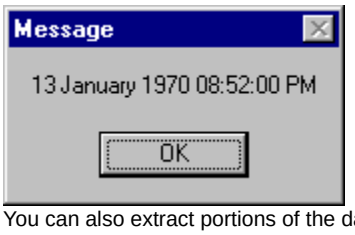
```
DIM dateAndTime AS DATE
dateAndTime = 34936.504
CURRDATE = dateAndTime
MESSAGE dateAndTime
```



To assign the system date and time to a variable:

The following example sets variable **dateNow** to your system's current date and time. A message box is used to display the date value of **dateNow**.

```
dim dateNow as date
dateNow = CURRDATE
MESSAGE dateNow
```



You can also extract portions of the date and convert them to strings and numbers. The following continues the above example:

```
myDateString$ = dateNow               ' create a string
myDay% = VAL(LEFT(myDateString$, 2)) ' extract the day as an integer
```

In the above example, the VAL and LEFT functions are used to extract an integer from the **myDateString** string variable. The method to extract portions from a date string differ depending on your Windows date settings.

Using CURRDATE for file date stamping:

The following example assigns the system date to the variable **xDate**. The third line sets the system date to February 26, 1994, and is then followed by the opening of a text file which will be stamped with the new system date. The last line resets the system date to its orignal value.

```
DIM xDate AS DATE
xDate = CURRDATE
CURRDATE = "02/26/1994"   'sets the system
OPEN "c:\log.txt" FOR OUTPUT AS 2
```

```
CURRDATE = xDate
```

Note

- Date and time formatting is set in your Windows Regional Setting. The Corel SCRIPT date format style is based on the Windows Short date style, and the time style is based on the Windows time style. To change your Windows settings, click Start, Settings, Control Panel. Double-click Regional Settings and choose the folders you want to change.

{button ,AL(`mid;left;val;cs_date_time;;',0,"Defaultoverview",)} **Related Topics**

WAIT FOR

Pauses script execution for a specified number of seconds.

Syntax
WAIT FOR x

Argument	Definition
x	Specifies the number of seconds to pause script execution. The parameter **x** can be a number, numeric variable or constant, or a numeric expression, but must be positive.

Example

```
WAIT FOR 3
```

The above example pauses script execution for three seconds.

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

WAIT UNTIL

Pauses script execution until the system timer matches a specified date serial number.

You should consider using the WAIT UNTIL statement for running scripts in off-peak times. For example, if you had a large print job to execute, you could use a script to run it during an off-peak period such as during the night.

Syntax
WAIT UNTIL x

Argument	Definition
x	Specifies the date serial number, which indicates when to resume execution. The parameter **x** can be a number, numeric variable or constant, or a numeric expression, but must be positive and greater than the system's current date.

Example

The following example pauses script execution until the system time matches 34936.25 (August 25, 1996, 6:00 A.M.).

```
DIM offPeak AS DATE
offPeak = 34936.25
WAIT UNTIL offPeak
```

The following example pauses script execution until 3:00 A.M. regardless of the date:

```
DIM today, tonite  AS DATE
DIM daypart AS LONG
today = CURRDATE
daypart = INT(today)  'daypart is set to 12:00 AM today
tonite = datepart + 1.125 'sets the date and time to 3 AM the next day
WAIT UNTIL tonite
```

By adding a day and 3 hours (1.125) to **datepart**, **tonite** is set to 3 AM tomorrow morning.

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

Currently the following Corel applications support Corel SCRIPT:

Application	WITHOBJECT string	WITHOBJECT constant*
CorelCAD 1.0	"CorelCad.automation.1"	
CorelDRAW 6.0	"CorelDraw.automation.6"	DRAW
CorelFLOW 3.0	"CorelFlow.automation.3"	FLOW
Corel PHOTO-PAINT 6.0	"CorelPhotoPaint.automation.6"	PAINT

***** When using the WITHOBJECT command, you have the option of using the string or a constant to specify a Corel application. For example:

WITHOBJECT "CorelDraw.automation.6" or **WITHOBJECT DRAW**

Currently the following Corel applications support OLE automation:

Application	OLE automation object name
CorelCAD 1.0	CorelCad.automation.1
CorelDRAW 6.0	CorelDraw.automation.6
CorelFLOW 3.0	CorelFlow.automation.3
Corel PHOTO-PAINT 6.0	CorelPhotoPaint.automation.6

Corel SCRIPT character map

Corel SCRIPT uses the American National Standards Institute (ANSI) character mapping. At 256 characters, the ANSI character set offers twice as many characters as ASCII text, including special characters such as the copyright symbol, accents, and mathematical and currency symbols.

The first 128 characters (numbers 0-127) in the ANSI set correspond to characters on a standard U.S. keyboard, and include many non-printing characters. The second 128 characters (numbers 128-255) return different characters, depending on the font used.

Because some character symbols in Corel SCRIPT are reserved for use in syntax, such as quotation marks for defining strings, the CHR function and ANSI character codes should be used to include special symbols in a script. For example, to add double quotation marks to a string variable, you use ANSI character 34:

```
s$ = CHR(34) + "This will be in double quotes." + CHR(34)
MESSAGE s$
```

You can also use the function to add a return and a line feed within a string, with character 13 and 10, respectively:

```
s$ = "String 1" + CHR(13) + CHR(10) + "String 2"
MESSAGE s$
```

The result will be the two strings on separate lines, as displayed in the message box.

Character Set (ANSI 0 - 127)

Character Set (ANSI 128 - 255)

{button ,AL(`using_strings;chr;cs_character_map;;;',0,"Defaultoverview",)} Related Topics

Character Set (ANSI 0 - 127)

0	☒	32	[SPACE]	64	@	96	`
1	☒	33	!	65	A	97	a
2	☒	34	"	66	B	98	b
3	☒	35	#	67	C	99	c
4	☒	36	$	68	D	100	d
5	☒	37	%	69	E	101	e
6	☒	38	&	70	F	102	f
7	☒	39	'	71	G	103	g
8	[BACKSPACE]	40	(	72	H	104	h
9	[TAB]	41	)	73	I	105	i
10	[LINEFEED]	42	*	74	J	106	j
11	▪	43	+	75	K	107	k
12	▪	44	,	76	L	108	l
13	[RETURN]	45	-	77	M	109	m
14	▪	46	.	78	N	110	n
15	▪	47	/	79	O	111	o
16	▪	48	0	80	P	112	p
17	▪	49	1	81	Q	113	q
18	▪	50	2	82	R	114	r
19	▪	51	3	83	S	115	s
20	▪	52	4	84	T	116	t
21	▪	53	5	85	U	117	u
22	▪	54	6	86	V	118	v
23	▪	55	7	87	W	119	w
24	▪	56	8	88	X	120	x
25	▪	57	9	89	Y	121	y
26	▪	58	:	90	Z	122	z
27	▪	59	;	91	[	123	{
28	▪	60	<	92	\	124	|
29	▪	61	=	93	]	125	}
30	▪	62	>	94	^	126	~
31	▪	63	?	95	_	127	▪

Note

- ▪ Characters displayed with a
- ▪ do not have a graphical representation in ANSI.

Character Set (ANSI 128 - 255)

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

Character Set (ANSI 128 - 255)

128	▪	160	[SPACE]	192	À	224	à
129	▪	161	¡	193	Á	225	á
130	▪	162	¢	194	Â	226	â
131	▪	163	£	195	Ã	227	ã
132	▪	164	¤	196	Ä	228	ä
133	▪	165	¥	197	Å	229	å
134	▪	166	¦	198	Æ	230	æ
135	▪	167	§	199	Ç	231	ç
136	▪	168	¨	200	È	232	è
137	▪	169	©	201	É	233	é
138	▪	170	ª	202	Ê	234	ê
139	▪	171	«	203	Ë	235	ë
140	▪	172	¬	204	Ì	236	ì
141	▪	173		205	Í	237	í
142	▪	174	®	206	Î	238	î
143	▪	175	▪	207	Ï	239	ï
144	▪	176	°	208	Ð	240	ð
145	▪	177	±	209	Ñ	241	ñ
146	▪	178	²	210	Ò	242	ò
147	▪	179	³	211	Ó	243	ó
148	▪	180	´	212	Ô	244	ô
149	▪	181	µ	213	Õ	245	õ
150	▪	182	¶	214	Ö	246	ö
151	▪	183	·	215	×	247	÷
152	▪	184	¸	216	Ø	248	ø
153	▪	185	¹	217	Ù	249	ù
154	▪	186	º	218	Ú	250	ú
155	▪	187	»	219	Û	251	û
156	▪	188	¼	220	Ü	252	ü
157	▪	189	½	221	Ý	253	ý
158	▪	190	¾	222	Þ	254	þ
159	▪	191	¿	223	ß	255	ÿ

Note

- ▪ Characters displayed with a ▪ do not have a graphical representation in ANSI.

Character Set (ANSI 0 - 127)

{button ,AL(`using_strings;chr;cs_character_map;;;',0,"Defaultoverview",)} Related Topics

ANGLECONVERT

Converts a number from one angle measurement to another.

Syntax

ANGLECONVERT(x{!|#}, **y**{#|!}, **z**{%|&|!|#|@}**)**

Argument	Definition
x{#|!}	Any constant from 1 to 5, that indicates the unit of measurement from which to convert. 1 = degrees 2 = radians 3 = gradients 4 = PAINT degrees (tenths of a degree) 5 = DRAW degrees (millionths of a degree)
y{#|!}	Any constant from 1 to 5, that indicates the unit of measurement to convert to. 1 = degrees 2 = radians 3 = gradients 4 = PAINT degrees (tenths of a degree) 5 = DRAW degrees (millionths of a degree)
z{%|&|!|#|@}	Any number, numeric variable or constant, or numeric expression.

Example

```
degs2rads = ANGLECONVERT(1, 2, 90)
```

The above example converts 90 degrees to radians and **x** equals 1.57142857142932.

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

LENGTHCONVERT

Converts a number from one length measurement to another.

Syntax

LENGTHCONVERT(x{#|!}, **y**{#|!}, **z**{%|&|!|#|@}**)**

Argument	Definition
x{#|!}	Any constant from 1 to 7, that indicates the unit of measurement from which to convert. 1 = inches 2 = centimeters 3 = points 4 = ciceros 5 = didots 6 = picas 7 = DRAW units (tenths of a micron)
y{#|!}	Any constant from 1 to 7, that indicates the unit of measurement to convert to. 1 = inches 2 = centimeters 3 = points 4 = ciceros 5 = didots 6 = picas 7 = DRAW units (tenths of a micron)
z{%|&|!|#|@}	Any number, numeric variable or constant, or numeric expression.

Note

- Tenths of a micron are used as the basic unit of measurement in Corel applications such as CorelDRAW.

Example

```
x = LENGTHCONVERT(1, 7, 1)
```

The above example converts one inch to tenths of a micron and **x** equals 254,000.

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

FROMINCHES

Converts the argument from inches to tenths of a micron.

Syntax

FROMINCHES(x{%|&|!|#|@**})**

Argument	Definition
x{%|&|!|#|@}	Any number, numeric variable or constant, or a numeric expression; specifies the argument to convert.

Note

- Tenths of a micron are used as the basic unit of measurement in Corel applications such as CorelDRAW.

Example

```
.CreateRectangle FROMINCHES(3), FROMINCHES(-2), FROMINCHES(0), FROMINCHES(1), FROMINCHES(0.25)
```

This CorelDRAW command would create a rectangle 3 by 3 inches. The rectangle's top left coordinate is -2, 3 inches relative to the origin, and the corners are 0.25 inches in diameter.

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

FROMDIDOTS

Converts the argument from didots to tenths of a micron.

Syntax
FROMDIDOTS(x{%|&|!|#|@}**)**

Argument	Definition
x{%|&|!|#|@}	Any number, numeric variable or constant, or a numeric expression; specifies the argument to convert.

Note

- Tenths of a micron are used as the basic unit of measurement in Corel applications such as CorelDRAW.

Example

```
.CreateRectangle FROMDIDOTS(50), FROMDIDOTS(-70), FROMDIDOTS(0), FROMDIDOTS(30),
FROMDIDOTS(20)
```

This CorelDRAW command would create a rectangle 100 by 50 didots. The rectangle's top left coordinate is -70, 50 didots relative to the center of the page, and the corners are 20 didots in diameter.

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

FROMPICAS

Converts the argument from picas to tenths of a micron.

Syntax

FROMPICAS(x{%|&|!|#|@}**)**

Argument	Definition
x{%|&|!|#|@}	Any number, numeric variable or constant, or a numeric expression; specifies the argument to convert.

Note

- Tenths of a micron are used as the basic unit of measurement in Corel applications such as CorelDRAW.

Example

```
.CreateRectangle FROMPICAS(18), FROMPICAS(-12), FROMPICAS(8), FROMPICAS(6), FROMPICAS(2)
```

This CorelDRAW command would create a rectangle 18 by 10 picas. The rectangle's top left coordinate is -12, 18 picas relative to the center of the page, and the corners are 2 picas in diameter.

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

FROMCICEROS

Converts the argument from ciceros to tenths of a micron.

Syntax

FROMCICEROS(x{%|&|!|#|@}**)**

Argument	Definition
x{%|&|!|#|@}	Any number, numeric variable or constant, or a numeric expression; specifies the argument to convert.

Note

- Tenths of a micron are used as the basic unit of measurement in Corel applications such as CorelDRAW.

Example

```
.CreateRectangle FROMCICEROS(18), FROMCICEROS(-12), FROMCICEROS(8), FROMCICEROS(6),
FROMCICEROS(1.5)
```

This CorelDRAW command would create a rectangle 18 by 10 ciceros. The rectangle's top left corner coordinate is -12, 18 ciceros relative to the center of the page, and the corners are 1.5 ciceros in diameter.

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

FROMPOINTS

Converts the argument from points to tenths of a micron.

Syntax

FROMPOINTS(x{%|&|!|#|@}**)**

Argument	Definition
x{%|&|!|#|@}	Any number, numeric variable or constant, or a numeric expression; specifies the argument to convert.

Note

- Tenths of a micron are used as the basic unit of measurement in Corel applications such as CorelDRAW.

Example

```
.CreateRectangle FROMPOINTS(210), FROMPOINTS(-140), FROMPOINTS(90), FROMPOINTS(70),
FROMPOINTS(1.75)
```

This CorelDRAW command would create a rectangle 210 by 140 points. The rectangle's top left corner coordinate is -140, 210 points relative to the center of the page, and the corners are 1.75 points in diameter.

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

FROMCENTIMETERS

Converts the argument from centimeters to tenths of a micron.

Syntax

FROMCENTIMETERS(x{%|&|!|#|@}**)**

Argument	Definition
x{%|&|!|#|@}	Any number, numeric variable or constant, or a numeric expression; specifies the argument to convert.

Note

- Tenths of a micron are used as the basic unit of measurement in Corel applications such as CorelDRAW.

Example

```
.CreateRectangle FROMCENTIMETERS(8), FROMCENTIMETERS(-5), FROMCENTIMETERS(0),
FROMCENTIMETERS(2.5), FROMCENTIMETERS(0.75)
```

This CorelDRAW command would create a rectangle 7.5 by 8 centimeters. The rectangle's top left corner coordinate is -5, 8 centimeters relative to the center of the page, and the corners are 0.75 centimeters in diameter.

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

TOINCHES

Converts the argument from tenths of a micron to inches.

Syntax
TOINCHES(x{%|&|!|#|@}**)**

Argument	Definition
x{%|&|!|#|@}	Any number, numeric variable or constant, or a numeric expression; specifies the argument to convert.

Note

- Tenths of a micron are used as the basic unit of measurement in Corel applications such as CorelDRAW.

Example

```
.GetPosition (x,y)
xInch = TOINCHES (x)
yInch = TOINCHES (y)
```

In this CorelDRAW example, **xInch** and **yInch** are set to the X and Y coordinates of the selected object in inches.

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

TODIDOTS

Converts the argument from tenths of a micron to didots.

Syntax
TODIDOTS(x{%|&|!|#|@}**)**

Argument	Definition
x{%|&|!|#|@}	Any number, numeric variable or constant, or a numeric expression; specifies the argument to convert.

Note

- Tenths of a micron are used as the basic unit of measurement in Corel applications such as CorelDRAW.

Example

```
.GetPosition (x,y)
xDidots = TODIDOTS (x)
yDidots = TODIDOTS (y)
```

In this CorelDRAW example, **xDidots** and **yDidots** are set to the X and Y coordinates of the selected object in didots.

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

TOPICAS

Converts the argument from tenths of a micron to picas.

Syntax
TOPICAS(x{%|&|!|#|@}**)**

Argument	Definition
x{%|&|!|#|@}	Any number, numeric variable or constant, or a numeric expression; specifies the argument to convert.

Note

- Tenths of a micron are used as the basic unit of measurement in Corel applications such as CorelDRAW.

Example

```
.GetPosition (x,y)
xPica = TOPICAS (x)
yPica = TOPICAS (y)
```

In this CorelDRAW example, **xPica** and **yPica** are set to the X and Y coordinates of the selected object in picas.

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

TOCICEROS

Converts the argument from tenths of a micron to ciceros.

Syntax
TOCICEROS(x{%|&|!|#|@}**)**

Argument	Definition
x{%|&|!|#|@}	Any number, numeric variable or constant, or a numeric expression; specifies the argument to convert.

Note

- Tenths of a micron are used as the basic unit of measurement in Corel applications such as CorelDRAW.

Example

```
.GetPosition (x,y)
xCiceros = TOCICEROS (x)
yCiceros = TOCICEROS (y)
```

In this CorelDRAW example, **xCiceros** and **yCiceros** are set to the X and Y coordinates of the selected object in ciceros.

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

TOPOINTS

Converts the argument from tenths of a micron to points.

Syntax
TOPOINTS(x{%|&|!|#|@}**)**

Argument	Definition
x{%|&|!|#|@}	Any number, numeric variable or constant, or a numeric expression; specifies the argument to convert.

Note

- Tenths of a micron are used as the basic unit of measurement in Corel applications such as CorelDRAW.

Example

```
.GetPosition (x,y)
xPoint = TOPOINTS (x)
yPoint = TOPOINTS (y)
```

In this CorelDRAW example, **xPoint** and **yPoint** are set to the X and Y coordinates of the selected object in points.

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

TOCENTIMETERS

Converts the argument from tenths of a micron to centimeters.

Syntax
TOCENTIMETERS(x{%|&|!|#|@}**)**

Argument	Definition
x{%|&|!|#|@}	Any number, numeric variable or constant, or a numeric expression; specifies the argument to convert.

Note

- Tenths of a micron are used as the basic unit of measurement in Corel applications such as CorelDRAW.

Example

```
.GetPosition (x,y)
xCM = TOCENTIMETERS (x)
yCM = TOCENTIMETERS (y)
```

In this CorelDRAW example, **xCM** and **yCM** are set to the X and Y coordinates of the selected object in centimeters.

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

To move a toolbar

1. Click the border of the toolbar.
2. Drag it to its new location.
   Right-click to cancel the movement.

Tip

- Double-click a toolbar's title or border to automatically dock and undock it.

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

To resize a toolbar

1. Move the cursor to the edge of a floating toolbar.
2. Drag the edge until the toolbar is the correct size.
   Right-click to cancel the movement.

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

To display and hide an existing toolbar

1. Click View, Toolbars.
2. Click the checkbox next to the toolbar you want to display or hide.

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

To assign a Corel SCRIPT script to an accelerator (shortcut) key

From your Corel application:

1. Click Tools, Customize.
2. Click Keyboard.
3. In the Commands box, double-click the Application Scripts folder or the General Scripts folder.

   The Application Scripts folder contains scripts that send instructions to the Corel application you are customizing. The General Scripts folder contains scripts that are not application specific. Based on a typical Corel installation, application scripts reside in the **C:\corel...\a*pplication*** folder, where ***application*** refers to the application's folder. For example, CorelDRAW scripts are saved in the **C:\corel60\draw** folder. General scripts normally reside in the **C:\corel...\scripts** folder.
4. Click the script.
5. Click the Press new shortcut key box.
6. Press the keyboard combination you want to assign to the command. To make a correction, press the Backspace key.

   You can have up to four layers of keystrokes. For example, the key combination CTRL+ALT+1,2,3,4 is accomplished by holding down the CTRL and ALT keys, then pressing the 1,2,3, and 4 keys in succession.

Note

- To have accelerator conflicts resolved automatically, enable Go to conflict on assign.

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

To assign a Corel SCRIPT script to a menu

From your Corel application:

1. Click Tools, Customize.
2. Click Menu.
3. In the Commands box, double-click the Application Scripts folder or the General Scripts folder.

   The Application Scripts folder contains scripts that send instructions to the Corel application you are customizing. The General Scripts folder contains scripts that are not application specific. Based on a typical Corel installation, application scripts reside in the **C:\corel...\a*pplication*** folder, where ***application*** refers to the application's folder. For example, CorelDRAW scripts are saved in the **C:\corel60\draw** folder. General scripts normally reside in the **C:\corel...\scripts** folder.
4. Click the script.
5. In the Menu box, click the menu or sub-menu where you want to add the command.
6. Click Add.

Tip

- Use the Separator button to add organizational lines to your menus.

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

To assign a Corel SCRIPT script to a toolbar button

From your Corel application:

1. Activate the toolbar you want to edit.
2. Click Tools, Customize.
3. In the Commands box, double-click the Application Scripts folder or the General Scripts folder.

   The Application Scripts folder contains scripts that send instructions to the Corel application you are customizing. The General Scripts folder contains scripts that are not application specific. Based on a typical Corel installation, application scripts reside in the **C:\corel...\a*pplication*** folder, where ***application*** refers to the application's folder. For example, CorelDRAW scripts are saved in the **C:\corel60\draw** folder. General scripts normally reside in the **C:\corel...\scripts** folder.
4. Click the script.
5. Drag the appropriate command button to the toolbar. Right-click to cancel the movement.

Tip

- If a script's first line, second line, or both are REM statements, they are displayed in the Description text box.

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

To run a Corel SCRIPT script from a Corel application

1. Click Tools:

   For **CorelCAD 1**: click Run Script.

   For **CorelDRAW 6**: click Scripts, Run.

   For **CorelFLOW 3**: click Run Script.

   For **Corel PHOTO-PAINT 6**: click Scripts, Run.
2. If the Corel SCRIPT script is not in the default folder, chose the drive and folder where it is stored.
3. Double-click the Corel SCRIPT script you want to run.

Notes

- You can use wild cards (* and ?) if you're not sure of the name of the file you want to run. For example, typing **script*.csc** in the File Name box and clicking OK lists all CSC files in the selected folder beginning with **script**. Typing **sc?.csc** in the File Name box and clicking OK lists all CSC files in the selected folder that begin with **sc** and are followed by only one more character.
- To run a script from your Corel application, the script must contain the WITHOBJECT construct.
- You can't undo executed script commands. If you're not sure whether a script is running to your specifications, then before the script is executed, save the documents in the Corel application that will receive the script instructions .
- You can terminate a script's execution by pressing ESC. In some cases, you may have to press ESC several times to terminate execution.

{button ,AL(`app_Corel_SCRIPT;script_files;ht_play_script_cse;;;',0,"Defaultoverview",)} Related Topics

Reserved keywords

The following keywords are reserved by CorelSCRIPT and cannot be used as variable, constant, procedure, function, subroutine, parameter, or array names:

ABS
ACOS
ALIAS
AND
ANGLECONVERT
APPEND
AS
ASC
ASIN
AT
ATAN

BEEP
BEGIN
BOOLEAN
BYREF
BYVAL

CALL
CANCELBUTTON
CASE
CBOL

CCUR
CDAT
CDBL
CHECKBOX
CHR
CINT
CLNG
CLOSE
COMBOBOX
CONST
COPY
COS
CSGN
CSTR
CURRDATE
CURRENCY
CURRFOLDER

DATA
DATE
DDCOMBOBOX
DDLISTBOX
DEC
DECLARE
DIALOG
DIM
DO
DOUBLE
DRAW

ELSE
ELSEIF
END
ENDIF
EOF
EQV
ERROR
EXIT
EXP

FAIL
FIELD
FILEATTR
FILEDATE
FILEMODE
FILEPOS
FILESIZE
FINDFIRSTFOLDER
FINDNEXTFOLDER
FIX

FLOW
FOR
FREEFILE
FROMCENTIMETERS
FROMCICEROS
FROMDIDOTS
FROMINCHES
FROMPICAS
FROMPOINTS
FUNCTION

GET
GETAPPHANDLE
GETFILEBOX
GETWINHANDLE
GLOBAL
GOSUB
GOTO
GROUPBOX

HELPBUTTON
HEX

IF
IMAGE
IMAGELISTBOX
IMP
INPUT
INPUTBOX
INSTR
INT
INTEGER
IS

KILL

LBOUND
LCASE
LEFT
LEN
LENGTHCONVERT
LET
LIB
LINE
LISTBOX
LN
LOCK
LOF
LOG
LONG

LOOP
LSET
LTRIM

MESSAGE
MESSAGEBOX
MID
MKFOLDER
MOD

NEXT
NOT

OBJECT
OKBUTTON
ON
OPEN
OPTIONBUTTON
OPTIONGROUP
OR
OUTPUT

PAINT
PRINT
PUSHBUTTON
PUT

RANDOMIZE
READ
REDIM
REM
RENAME
RESET
RESTORE
RESUME
RIGHT
RMFOLDER
RND
RTRIM

SEEK
SELECT
SET
SGN
SIN
SINGLE
SPACE
SPINCONTROL
SQR
STATIC

STEP
STOP
STRING
STR
SUB
SWAP

TAN
TEXT
TEXTBOX
THEN
TIMER
TO
TOCENTIMETERS
TOCICEROS
TODIDOTS
TOINCHES
TOPICAS
TOPOINTS
TYPE

UBOUND
UCASE
UNLOCK
UNTIL

VAL
VARIANT

WAIT
WEND
WHILE
WITH
WITHOBJECT
WRITE

XOR

This word is reserved for future use.

Trappable error codes

If errors occur when you execute a script, an error message is displayed in the Compiler Output window. If you're using an error-handling routine, the Corel SCRIPT global variable ERRNUM is assigned a value based on the error. For more information about ERRNUM, see the ONERROR statement.

ERRNUM Value	Error Message
1	Internal Error
2	Unknown error
3	Variable is not initialized
50	Division by 0
51	Type mismatch
52	Out of memory
53	Overflow
54	Invalid array index
55	Invalid specification of array dimensions
56	Resume without error
200	Invalid file number
201	Unable to open file
202	File handle out of range
203	Error writing to file
204	Error reading file
205	Invalid file position
250	Unable to change folder
251	Unable to make folder
252	Unable to remove folder
253	Unable to delete file
300	Incorrect parameter count
301	Unable to get DLL entry point
302	Unable to execute function
303	Object not initialized through WITHOBJECT
304	Invalid parameter
350	Unable to initialize dialog
351	Invalid dialog
600	OLE Automation Error
601	Invalid number of parameters
602	Bad variable type
603	Object does not support this function
604	Object does not support named arguments
605	Value out of range
606	Invalid Parameter name
607	Argument Type mismatch
608	Unknown Interface

609	Unknown language
610	Unspecified parameter is not optional
800 to 999	User defined errors numbers.
1000 +	Corel application errors

Note

- Not all Corel SCRIPT errors are trappable.

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

Trappable Error: 1

ERRNUM 1 indicates an internal error.

Possible causes:

- Your system resources may be low.
- You may be low on hard disk space.

Possible solutions:

- Free up system resources and hard disk space.

Trappable Error: 2

ERRNUM 2 indicates an unknown error.

Possible causes:

- Your system resources may be low.
- You may be low on hard disk space.

Possible solutions:

- Free up system resources and hard disk space.

Trappable Error: 3

ERRNUM 3 indicates an uninitialized variable.

Possible causes:

- You may have misspelled a variable name.
- You may not have initialized the variable.

Possible solutions:

- Confirm the spelling of the variables in your script.
- Confirm that all variables are initialized.

Trappable Error: 50

ERRNUM 50 indicates an attempt to divide by zero.

Possible causes:

- You may not have initialized the variable.
- You may have misspelled a variable being used as a denominator.

Possible solutions:

- Ensure that all variables are initialized and spelled correctly.

Trappable Error: 51

ERRNUM 51 indicates a type mismatch.

Possible causes:

- You may be mis-assigning a value to a variable.
- You may have misspelled the variable name.

Possible solutions:

- Confirm that all variables are spelled correctly.
- Confirm that you are assigning values to the correct variables.

Trappable Error: 52

ERRNUM 52 indicates that the system is out of memory.

Possible causes:

- You may have an infinite loop in your script.
- You may have an over-sized array.

Possible solutions:

- Check the logic in your script, ensuring that infinite loops are removed.
- Ensure that all arrays are assigned a reasonable size.

Trappable Error: 54

ERRNUM 54 indicates an invalid array index.

Possible causes:

- You may not have correctly initialized your index or your array.
- Your logic might be incorrectly adding to your index.

Possible solutions:

- Ensure that your array and index are initialized.
- Check the logic in your script, confirming all modifications to the array index.

Trappable Error: 53

ERRNUM 53 indicates an overflow.

Possible causes:

- You may have assigned an inappropriate value to a variable type, such as assigning a number greater than 32 767 to an integer.

Possible solutions:

- Go through your script's logic, ensuring that all values are appropriate.
- Confirm your variable declarations to ensure than each variable is of the correct data type.

Trappable Error: 55

ERRNUM 55 indicates an invalid specification of array dimensions.

Possible causes:

- The number of dimensions doesn't match the definition in the DIM statement.
- Invalid specification of an array dimension such as 7 to -3.
- Invalid specification of the dimension in the LBOUND or UBOUND statement.

Possible solutions:

- Correct the DIM, LBOUND, or UBOUND statement.

Trappable Error: 56

ERRNUM 56 indicates a resume without error error.

Possible causes:

- Your script may have executed a RESUME statement outside of an active error-handling routine.

Possible solutions:

- You should place a STOP, EXIT FUNCTION, or an EXIT SUB before an error-handling routine in a script to prevent it from being executed when no error has occurred.

Trappable Error: 200

ERRNUM 200 indicates an invalid file number.

Possible causes:

- You may be assigning a file number outside the range of one to ten.

Possible solutions:

- Confirm that your file numbers do not exceed ten.

Trappable Error: 201

ERRNUM 201 indicates an inability to open a file.

Possible causes:

- You may not be allowed access to that file.
- The file may not be present.
- The file may be corrupted.
- The file may not be of the correct file-type.
- The file may already be in use.

Possible solutions:

- Confirm that you are allowed to access the file.
- Confirm that the file is both present and of the correct file-type.
- Confirm that the file is not corrupted.
- Confirm that the file is not already in use.

Trappable Error: 202

ERRNUM 202 indicates an out-of-range file handle.

Possible causes:

- You may be assigning a file number outside the range of one to ten.

Possible solutions:

- Confirm that your file number is between one and ten. FREEFILE can help you find an unused file handle.

Trappable Error: 203

ERRNUM 203 indicates an error writing to a file.

Possible causes:

- You may not be allowed access to that file.
- The file may not be present.
- The file may be corrupted.
- The file may not be of the correct file-type.
- The file may already be in use.

Possible solutions:

- Confirm that you are allowed to access the file.
- Confirm that the file is both present and of the correct file-type.
- Confirm that the file is not corrupted, and is not already in use.

Trappable Error: 204

ERRNUM 204 indicates an error reading a file.

Possible causes:

- You may not be allowed access to that file.
- The file may not be present.
- The file may be corrupted.
- The file may not be of the correct file-type.
- The file may already be in use.

Possible solutions:

- Confirm that you are allowed to access the file.
- Confirm that the file is both present and of the correct file-type.
- Confirm that the file is not corrupted.
- Confirm that the file is not already in use.

Trappable Error: 205

ERRNUM 205 indicates an invalid file position.

Possible causes:

- You are trying to access an area outside the file.

Possible solutions:

- Use the LOF function to determine the file length, and access a position within that area.

Trappable Error: 250

ERRNUM 250 indicates an error in changing files.

Possible causes:

- You may not be allowed access to that file.
- The file may not be present.
- The file may be corrupted.
- The file may not be of the correct file-type.
- The file may already be in use.

Possible solutions:

- Confirm that you are allowed to access the file.
- Confirm that the file is both present and of the correct file-type.
- Confirm that the file is not corrupted, and is not already in use.

Trappable Error: 251

ERRNUM 251 indicates an error in creating a folder.

Possible causes:

- You may not have file-creation privileges.
- The filename may already be present.

Possible solutions:

- Confirm that you are allowed to create files.
- Confirm that the file is not already in use.

Trappable Error: 252

ERRNUM 252 indicates an error in removing a folder.

Possible causes:

- You may not be allowed to remove that file.
- The file may not be present.
- The file may already be in use.

Possible solutions:

- Confirm that you are allowed to access the file.
- Confirm that the file is present.
- Confirm that the file is not already in use.

Trappable Error: 253

ERRNUM 253 indicates an error in deleting a file.

Possible causes:

- You may not be allowed access to that file.
- The file may not be present.
- The file may already be in use.

Possible solutions:

- Confirm that you are allowed to access the file.
- Confirm that the file is present.
- Confirm that the file is not already in use.

Trappable Error: 300

ERRNUM 300 indicates an incorrect parameter count.

Possible causes:

- You may have omitted a parameter, or provided an unnecessary argument.

Possible solutions:

- Confirm that your parameter count is correct throughout the script.

Trappable Error: 301

ERRNUM 301 indicates a problem in getting a Dynamic Linked Library entry point.

Possible causes:

- The DLL does not exist.
- The function does not exist in the DLL.
- You may have misspelled the function name, or the case may not match.

Possible solutions:

- Check that the DLL exists, and that the function is applicable to that DLL.
- Confirm the spelling and case of the function name.

Trappable Error: 302

ERRNUM 302 indicates an error in executing a function.

Possible causes:

- Your system resources may be low.
- You may be low on hard disk space.

Possible solutions:

- Free up system resources and hard disk space.

Trappable Error: 303

ERRNUM 303 indicates an object which hasn't been initialized through WITHOBJECT.

Possible causes:

- The WITHOBJECT statement may have been by-passed by flow-control statements, such as GOTO.

Possible solutions:

- Step through the flow of the script, to confirm execution of the WITHOBJECT statement.

Trappable Error: 304

ERRNUM 304 indicates an invalid parameter.

Possible causes:

- You may have used parameter that is out of range.

Possible solutions:

- Confirm that the value of the parameters is in the acceptable range.
- Confirm data type of the parameters.

Trappable Error: 350

ERRNUM 350 indicates an error in initializing a dialog.

Possible causes:

- Your system resources may be low.
- You may be low on hard disk space.

Possible solutions:

- Free up system resources and hard disk space.

Trappable Error: 351

ERRNUM 351 indicates an invalid dialog.

Possible causes:

- Your dialog may contain invalid controls, such as an incorrectly placed button.

Possible solutions:

- Confirm that the dialog controls are positioned within reasonable bounds.

Trappable Error: 600

ERRNUM 600 indicates an error in OLE automation.

Possible causes:

- The application may have problems opening.
- The object name may be incorrect.

Possible solutions:

- Check the OLE automation errors.

Trappable Error: 601

ERRNUM 601 indicates an invalid number of parameters.

Possible causes:

- The application is expecting a different number of parameters.

Possible solutions:

- Check that all parameters are required, and that all required parameters are there.

Trappable Error: 602

ERRNUM 602 indicates a bad variable type.

Possible causes:

- You have passed the function a parameter of an unknown variable type.

Possible solutions:

- Confirm the variable’s data type.

Trappable Error: 603

ERRNUM 603 indicates that the object doesn’t support this function.

Possible causes:

- The object doesn’t support this function.

Possible solutions:

- Confirm that this function is supported by the object.

Trappable Error: 604

ERRNUM 604 indicates that the object doesn’t support the named arguments.

Possible causes:

- Some applications do not support named arguments.

Possible solutions:

- Confirm that the arguments are suitable for the application.

Trappable Error: 605

ERRNUM 605 indicates a value out of the acceptable range.

Possible causes:

- You may have passed a large value to an application which expected a smaller value.

Possible solutions:

- Confirm that the value you are trying to pass to the application is within an acceptable range, and is of a suitable data type.

Trappable Error: 606

ERRNUM 606 indicates an invalid parameter name.

Possible causes:

- You may have passed an incorrectly spelled named argument.

Possible solutions:

- Confirm that all named arguments are recognized.

Trappable Error: 607

ERRNUM 607 indicates an argument type mismatch.

Possible causes:

- You may have passed a variable which is not of the appropriate data type.

Possible solutions:

- Confirm that the data type is appropriate.

Trappable Error: 608

ERRNUM 608 indicates an unknown interface.

Possible causes:

- The object does not support OLE automation.

Possible solutions:

- Change the object, if necessary.

Trappable Error: 609

ERRNUM 609 indicates an unknown language..

Possible causes:

- The application is unable to recognize the international language of the functions.

Possible solutions:

- Confirm that the language is compatible.

Trappable Error: 610

ERRNUM 610 indicates that a required parameter is missing.

Possible causes:

- A required parameter is missing.

Possible solutions:

- Confirm that all required parameters are present.

Mathematical functions

You can use to the following formulas to create mathematical functions in Corel SCRIPT scripts. The formulas are based on Corel SCRIPT functions.

Function	To use this...	...paste this formula into your script
Secant	SEC(X) =	1 / COS(X)
Cosecant	COSEC(X) =	1 / SIN(X)
Cotangent	COTAN(X) =	1 / TAN(X)
Arc secant	ARCSEC(X) =	ACOS (1 / X)
Arc cosecant	ARCCOSEC(X) =	ASIN (1 / X)
Arc cotangent	ARCCOTAN(X) =	ATAN (1 / X)
Hyperbolic sine	HSIN(X) =	(EXP(X)- EXP (-X))/ 2
Hyperbolic cosine	HCOS(X) =	(EXP(X)+ EXP (-X))/ 2
Hyperbolic tangent	HTAN(X) =	(EXP(X)- EXP (-X)) / (EXP(X)+ EXP (-X))
Hyperbolic secant	HSEC(X) =	2 / (EXP(X)+ EXP (-X))
Hyperbolic cosecant	HCOSEC(X) =	2 / (EXP(X)- EXP (-X))
Hyperbolic cotangent	HCOTAN(X) =	(EXP(X)+ EXP (-X)) / (EXP(X)- EXP (-X))
Inverse hyperbolic sine	HARCSIN(X) =	LOG (X + SQR (X * X + 1))
Inverse hyperbolic cosine	HARCCOS(X) =	LOG (X + SQR (X * X -1))
Inverse hyperbolic tangent	HARCTAN(X) =	LOG ((1 + X)/(1 - X)) / 2
Inverse hyperbolic secant	HARCSEC(X) =	LOG ((SQR (-X * X + 1) + 1) / X
Inverse hyperbolic cosecant	HARCCOSEC(X)	LOG ((SQR (X) * SQR (X * X + 1) +1) / X)
Inverse hyperbolic cotangent	HARCCOTAN(X) =	LOG ((X + 1)/(X - 1)) / 2

Note

- You can copy a formula into a script by selecting it, clicking right-mouse button, Copy. The formula is placed on the clipboard ready to be pasted.

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

This button is not available in this application.

Resets a selected toolbar.

Displays the available toolbars. Enable the checkbox next to a toolbar to activate it.

This button is not available in this application.

Enables large toolbar buttons.

Enables medium toolbar buttons.

Enables small toolbar buttons.

Opens detailed online Help for this dialog box.

Customizing toolbars

You can use the toolbars dialog box to customize your applications toolbar settings. You can display and hide toolbars, and resize buttons.

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

Displays the available command categories. Click a category to display its command buttons.

Displays the command buttons for the current command category. Click a button to see its description, or drag it to add it to any toolbar on the screen.

A command button. Click it to see its description, or drag it to add it to any toolbar on the screen.

Gives a short description of any toolbar button you click.

Resets the toolbar assignments to their original configuration.

Displays the system information for the chosen category.

Opens the System Info dialog box where you can get information about your system, display, printing, Corel EXEs & DLLs and system DLLs.

Displays the disk space available on the drive where the program is installed.

Displays the name of the registered user and the serial number.

Displays copyright information.

Displays the version of the product currently installed.

Displays the name of the product.

Double-click to open the credits window. To exit the credits and return to your program, click the ESC button.

Provides a list of system information categories. Click one of the following:

System	Displays information about your computer For example, Windows version or processor.
Display	Displays information about your monitor. For example, driver,or driver version.
Printing	Displays information about installed printers.
Corel EXEs and DLLs	Displays information all of the Corel EXEs and DLLs.
System DLLs:	Displays all of the system DLLs.

Saves all system information as SYSINFO.TXT. Once it's saved, a message box appears informing you of the location of the saved file. Double-click to open the credits window. To exit the credits and return to your program, click the ESC button.

Double-click to open the credits window. To exit the credits and return to your program, click the ESC button.

Displays information about the contents of the file.

ANSI
Alignment, relative
Alignment, text
Anti-aliasing
Application command
Argument
Array
ASCII

Binary
Bitmap
BMP
Boolean variable
Breakpoint
Brighten
By reference
By value

Character code
Character set
Cicero
CMYK
Compile time
Constant
Cursor

Data type

Debug
Declaration, constant
Declaration, variable
Dialog box
DLL
Duotone

EOF

Folder
Font
Function

Icon
Identifier
Initialization
Intrinsic statement

Marquee
Maximize
Menu bar

Operators

Procedure

Return value
Run
Runtime

Script
Seed value
Subroutine
Syntax

Trace
Type-declaration character

Variable

Window

Alignment, relative

Setting two or more objects to some meaningful spatial relationship, such as centering them or distributing them evenly along a line.

Alignment, text

Text alignment affects text objects of more than one line. The lines of text may be aligned to the left edge, center, or right edge of the text frame.

ANSI

The American National Standards Institute character set. It consists of 256 characters; the first 128 are the same as the ASCII character set.

Anti-aliasing

The procedure of filtering an image to remove jagged edges from shapes. Anti-aliasing fills in the jagged pixels with intermediate colors (or shades of gray). This procedure smoothes transitions between colors.

Application command

A Corel application command. For example, the CorelDRAW command GetRectangle returns the coordinates of a rectangle.

Argument

A number, constant, or expression which is supplied for use in executing a procedure.

Array

An ordered set of items of the same data type. The items can be referred to both as a unit and individually.

ASCII

The American Standard Code for Information Interchange character set, which consists of the characters available on a standard 128 character keyboard, including non-printable control codes such as carriage returns and page breaks.

Binary

A system of counting based on a unit of two.

Bitmap

An image composed of a series of pixels or dots. Scanners and paint programs, such as Corel PHOTO-PAINT, generate this type of image. By contrast, CorelDRAW creates images using vector objects—shapes stored internally as mathematical equations.

BMP

The filename extension for Windows bitmap files.

Boolean variable

A variable with only 1 or 0 as possible values.

Breakpoint

A line in a script at which execution pauses, usually to aid in debugging. Breakpoints are removed before the script is used.

Brighten

Brighten increases the level of white or light in colors. The effect lightens the image or area.

By reference

A way of passing a variable to a function that allows you to alter the value.

By value

A way of passing a variable to a function that allows you to alter a copy of the variable but not change the value of the original variable.

Character code

The number corresponding to a character in a character set, such as the ASCII or ANSI character sets.

Character set

The letters, punctuation marks, and special characters in a font. Foreign language accents and mathematical symbols are examples of special characters.

Cicero

A unit of measure equivalent to 12 didots. Five point six, three (5.63) ciceros equals one inch.

CMYK

The subtractive color model used in printing. Colors are created by assembling different densities of cyan, magenta, yellow, and black pigments on a surface.

Compile time

The period of time during which source code is translated into executable code.

Constant

A name given to a particular value, such as pi (3.14152), or a name, such as *MyCompany*. A constant's value cannot be altered.

Cursor

Also called the mouse pointer. It indicates the object, command, tool, or other screen item you want to select. The shape of the cursor changes depending on the tool or command you have selected.

Data type

A definition of the type of information that can be stored in a variable or constant. For example, *integer* and *string* are both data types.

Debug

To remove the syntax and logic errors that prevent the script from either being compiled, or executing correctly.

Declaration, constant

A statement that states the name used to refer to a value throughout the script. For example, you would declare the constant *pi* to be *3.14159*.

Declaration, variable

A statement that states the name, data type, and initial value of a variable to be used throughout the script. For example, you would declare a variable *sum* to be an *integer*, with the initial value of *0*.

Dialog box

A window displayed when additional information is needed to perform an action.

DLL (Dynamically Linked Library)

A set of routines that can be included in the script file at run-time.

Duotone

A duotone is a grayscale image that has been converted into a two-color image. Using two colors of ink, instead of four, significantly reduces the costs of printing while still providing a wide range of colors to choose from. The duotone feature is ideal for adding an accent color to a photograph or for extending the tonal ranges of inks.

EOF (End Of File)

Exponential regression

A least squares approximation line of the form $y = a(b^x)$ for a selected series.

Expression

Any variable--positive, negative, fractional, whole numbers, or formulas.

Folder

Folders are used to store and organize your documents, programs, and other files. For example, you could create a folder called LOGOS for storing logo designs. In previous versions of Windows, folders were called directories.

Font

A set of characters in a given typeface and point size (see below). For example, 10 point Times Roman. Most fonts are available in families that include different weights or styles such as bold and italic.

10 pts 18 pts 36 pts

Function

A user-defined procedure, called from a script, that executes a series of commands and returns the result to the script.

Icon

A pictorial representation of a tool, object, file, or other program item. An item is selected by clicking once, or sometimes double-clicking, on its icon.

Identifier

A name used to refer to a particular constant or variable throughout the script.

Initialization

A statement declaring the initial value of a variable, which can also determine the variable's data type. Commonly, numeric variables are initialized to zero, and character strings are initialized to an empty string.

Integers

The number set consisting of all the positive whole numbers, their negatives, and zero. Integers are a subset of rational numbers.

Example: 10, -10, and 0 are integers.

Intrinsic statement

A CorelSCRIPT command that controls the logical progression of the script. An example of an intrinsic statement is the FUNCTION...ENDFUNCTION statement.

Irrational numbers

Numbers that can be expressed as infinite decimals with an indefinite sequence and that cannot be expressed as the quotient of two integers

Example: The square root of 2, which is 1.414213562373

Linear regression

A least squares approximation line of the formula y = a0+a1x for a selected series. This formula is equal to a polynomial fit with the Order set to 1. Linear regression is determined by the following formula:

$$E(Y|x) = b_0 + b_1 \times x$$

Variable X is an independent variable, Y is a dependent variable, $E(Y|x)$ is a conditional expectation of random variable Y for fixed value of the variable X. In practice, the parameters *0* and *1* of the linear regression are unknown, and their values evaluated through the Y and X variables.

Parameter *0* determines the intercept of the linear regression.

Parameter *1* determines the slope of the linear regression.

Marquee

When you drag diagonally to select a set of elements or region of an image, the rectangle created between the start point (mouse down) and the end point (mouse up) is called a marquee.

Maximize

To enlarge an application window to full-screen size.

Mean

A horizontal or vertical line drawn at the mean of the data points.

The "mean" equals the sum of the values divided by the number of values (e.g. [P1 + P2 + P3] / 3).

Menu bar

The bar near the top of the window that contains the names of the program menus.

Mixed reference

References are used in formulas to identify cells. **Mixed references** use one absolute co-ordinate and one relative co-ordinate (i.e., $E4). In this case, the column is absolute, which means that it will remain at column E if you move the cell that contains this reference. The row co-ordinate is relative and will be updated if such a change occurs.

Syntax: **(column letter.Row number) or (Column letter,$Row number)**

The $ identifies the absolute part of that reference; i.e., either the row number or column letter remain as they are when the cell containing the reference is moved.

See also Relative reference and Absolute reference.

Operators

Signs or symbols used to add, subtract, multiply, or divide (+, - , *, /, (),{}, and ^) inserted in formulas to activate various mathematical or logical operations.

Polynomial regression line

A line that curve fits to the data points following the polynomial regression formula: $y = a0 + a1x + a2x^2 + ... + anx^n$ where n = the order of the polynomial. Note that the order "0" draws a line at the mean of the data. This curved line is a least squares approximation of data for a selected series.

Power law regression

A line that fits the data points in accordance with the power law regression formula $y = ax^b$. This curved line is a least squares approximation of data for a selected series.

Procedure

A function or sub-routine.

Real numbers

The number set comprised of all the rational numbers and irrational numbers. Integers are a subset of the rational numbers.

Example: 25 and -10 are integers and real numbers because they are positive and negative whole numbers; 25.349 is a real number, but not an integer because it includes decimal places

Relative reference

References are used in formulas to indicate the position of cells in relation to the cell containing a formula. If you move or copy cells that use relative references, the references included in formulas are automatically updated to the new location of the referenced cell.

Syntax: **(Column letter, Row number)**

For example, a cell C10 contains a formula which includes the reference C5. If the formula is moved to cell D15, the reference will change so that it still points to the cell that is five cells to the left. The new reference will be D10.

See also Absolute reference and Mixed reference.

Return value

The value that is passed back to the function call.

Root

The root describes the highest level of the hierarchy, the universe. When you are Jumped Into a group, the group box is the highest level of the hierarchy, and, therefore, the local root.

Run

Execution of the script.

Runtime

The time during which the script executes.

Scale

A set of numbers attached to a numeric axis that indicates the values of the data. Also refers to changing the size of an object, or group of objects, while maintaining the original aspect ratio.

Script

A series of commands that can be executed by running the script. Scripts can be edited in the SCRIPT editor.

Section (in numeric format)

One of three possible components of a numeric format. The syntax for a numeric format is as follows:

Section_1;Section_2;Section_3

Each section is a series of codes that assign attributes to numbers. The attributes can include the font, size, style and color. Section_1 is the format for positive numbers, Section_2 for negative numbers, and Section_3 for zero values.

If a numeric format has only one section, the format is applicable to all numbers. If it has two sections, the first one is the format for positive and zero values and the second one is for negative numbers.

Seed value

The initial value from which a random series of numbers is generated. Typically, the system timer provides the seed value.

Subroutine

A user-defined procedure, called from a script, that executes a series of commands and does not return any information to the script.

Syntax

The set of rules that determine the order and format in which a command is made in order for it to be understood by the compiler.

Trace

To follow the progression of the script, line-by-line, as it executes.

Type-declaration character

The character suffixed to a variable name, indicating the data type of the variable.

Variable

A name that refers to data stored in a particular space in memory. The variable's data type determines the amount of space allocated to the variable. Variables can be changed during runtime.

Window

A rectangular area on the screen in which applications are displayed. Every application window has a title bar and menu bar along the top and one or two scroll bars along the sides or bottom.