Language Reference

REALbasic®

Create your own software.™

REALbasic Language Reference

Mailing Address	REAL Software, Inc. 1705 South Capital of Texas Highway Suite 310 Austin, TX 78746
Web Site	http://www.realsoftware.com
ftp Site	ftp://ftp.realsoftware.com
Support	REALbasic Feedback at the REAL Software web site.
Bugs/ Feature Requests	Submit via REALbasic Feedback at the REAL Software web site.
Database Plug-ins	REALbasic CD-ROM; the most current versions at www.realsoftware.com
Sales	sales@realsoftware.com
Phone	512-328-REAL (7325)
Fax	512-328-7372

Version 5.2, June, 2003.

Introduction

The Language Reference gives you fast access to information about all the REALbasic language elements: objects, keywords, operators, constants, error messages, runtime exceptions, built-in methods and functions, and so forth. Wherever applicable, the properties and methods belonging to an object are documented, along with cross-references to related objects.

There is also a Quickstart, a Tutorial, and User's Guide which teach REALbasic programming concepts. Your comments (both favorable and unfavorable) are welcome. Send them to docs@realsoftware.com.

Conventions

Property names in **bold** are read-only at runtime. This means that your code can read their values but cannot change their values.

Optional Parameters in syntax statements are surrounded by brackets- [ ].

Items in blue underline (in the PDF version of the Language Reference) are hypertext links to other parts of the document.

In some cases, code that is irrelevant to the example is omitted for sake of clarity. Omitted code is indicated by bullets.

The Class Hierarchy

Objects in REALbasic are arranged in a hierarchy. This means that an object inherits the properties, methods, and events of its super class (the classes above it in the hierarchy) and may contain properties, methods, and events that are unique to the object. The figure below shows the REALbasic class hierarchy. A PushButton for example, is a subclass of a RectControl, which is a subclass of a Control, which is a

subclass of an Object (the base class for all objects). Object classes in bold in the figure below are not directly accessible. You can only access their subclasses.

Cross-Platform Development

Using Target Flags

You can develop your application on the Macintosh platform and compile it for Windows. If your application will run on either Macintosh or Windows, you can use the following global Boolean constants to determine which type of code is currently executing.

Boolean	Description
Target68K	The compiled application is currently running 68K Macintosh code on a 68K machine or in emulation mode on a PPC machine. Obsolete and retained only for compatibility with previous versions of REALbasic. REALbasic versions 4.0 and above do not build 68K apps. Version 3.5.2 is the last version of REALbasic that builds 68K apps.
TargetMacOS	The compiled application is currently running Macintosh code, PPC 'classic', or Mac OS X.
TargetPPC	The compiled application is currently running PowerPC code within a 'classic' Mac OS.
TargetCarbon	The compiled application is currently running Carbon/Mac OS X code.
TargetWin32	The compiled application is currently running Win32 code.
DebugBuild	The application is running within the REALbasic application, i.e., from choosing Debug ▶ Run.

Conditional Compilation

If some of your code should be compiled only for a particular target platform (such as AppleScript-specific code or toolbox calls), you can use #If statements to compile that code only for the appropriate platform. Use the following structure:

```
#If TargetBoolean
 //OS-specific code
 //included in built app
 //only for target platform
#Else  //(optional)
 //(optional) Other OS-specific code
#Endif
```

Language Reference

The following is an alphabetical listing of the commands available in the REALbasic language.

- Operator

Used to find the difference between two numbers.

Syntax

result=expression1 - expression2

Part	Type	Description
result	Number	The difference between *expression1* and *expression2*.
expression1	Number	Any numeric expression.
expression2	Number	Any numeric expression.

Examples

This example stores the difference between two numbers in a variable:

```
Dim x as Integer
x=50-30   //x is 20
```

#If ...#Else...#Endif Statement

Used to control conditional compilation.

Syntax

#If *TargetBoolean* [Then]

//OS specific code

#Else

//Other OS-specific code

#Endif

Part	Type	Description
TargetBoolean	Boolean Constant	TargetMacOS, TargetPPC, Target68K, TargetCarbon, TargetWin32, or DebugBuild constant, used to determine the operating system that will include the code the follows.

Notes

Use conditional compilation to isolate platform-specific statements such as toolbox calls or AppleEvent routines. The code following the #If statement is included only in the build for that operating system. Target68K, TargetPPC, and TargetCarbon are mutually exclusive subsets of TargetMacOS. Please note that REALbasic no longer builds for 68K Macintoshes.

Example

The following example assigns values to the (user-defined) Separator property of the App class in its Open event handler. It will be used to specify full pathnames that are correct on both Macintosh and Windows.

```
#If TargetMacOS
  Separator=":"
#Endif
#If TargetWin32
  Separator="\"
#Endif
```

See Also

Declare statement, AppleEvent class; TargetMacOS, TargetPPC, Target68K, TargetCarbon, TargetWin32, DebugBuild constants.

#Pragma Directives

Used to speed up code execution by turning off certain automatic tasks. When a pragma directive is used, the automatic task remains off until the end of the method in which it is called (or is enabled by calling it again and specifying True), but not in other methods that might be called within the original method.

Syntax

#pragma *Directive*

or

#pragma *Directive* True or False

Directive	Description
BackgroundTasks	Enables or disables auto-yield to background threads. In addition to the pragma directive, specify True or False. Setting this directive to False is the same as using disableBackgroundTasks)
BoundsChecking	Enables or disables bounds checking. In addition to the pragma directive, specify True or False. Specifying False is the same as using disableBoundsChecking.
DisableAutoWaitCursor	Used to disable the automatic display of the wait cursor (or watch cursor). In versions 5.0 and above, the scope of this pragma is no longer global. The wait cursor will be disabled in the method that calls the pragma until the method ends. For example, if DisableAutoWaitCursor is called in a Pushbutton that runs a loop, the wait cursor will be disabled only until the loop runs and the Action event has completed.

Directive	Description
DisableBackgroundTasks	Used to turn off automatic background task handling for code after the #pragma. It prevents REALbasic from calling the WaitNextEvent toolbox routine, so it never yields time to any other applications. It, therefore, can speed up very processor-intensive operations but prevents REALbasic from displaying the Watch cursor, may halt normal background updating of interface elements, and prevents other threads from executing.
DisableBoundsChecking	Used to turn off array bounds checking on array index values in code after the #pragma. Works only for 1-dimensional boolean arrays.
NilObjectChecking	Controls whether to automatically check objects for Nil before accessing properties and calling methods. In addition to the pragma directive, specify True or False.
StackOverflowChecking	Controls whether to check for stack overflows. In addition to the pragma directive, specify True or False.

Notes

Four pragma directives require that you pass True or False to turn the option on or off. Since these pragmas can be enabled or disabled, you can now do things like disable background tasks for an inner loop, but leave them enabled for the outer loop:

```
For y = 0 To Height
  #pragma BackgroundTasks False
  For x = 0 To Width
    //...process some pixels...
  Next
  #pragma BackgroundTasks True
Next
```

& Operator

Used to represent hexadecimal, octal, and binary constants.

Syntax

&h prefix for hexadecimal numbers

&o prefix for octal numbers

&b for binary numbers

Examples

This example shows a hexadecimal constant:

```
&hA4
```

This example shows an octal constant:

```
&o527
```

This example shows a binary constant:

```
&b10010101
```

See Also Bin, Oct, Hex functions.

' Operator

Used to add comments to code.

Syntax **' text string**

Notes REALbasic's compiler will ignore any comments you enter following the ' operator. Comments are not included in stand-alone applications.

The // operator or Rem statement can be used to comment code as well. Comments can appear on separate lines or on the same line as executable code, provided they are to the right of any code that will execute. Valid comments appear in the Code Editor in red.

The Command-' keyboard shortcut toggles the selected lines of code between commented out and 'live' states. If the insertion point is somewhere in the middle of the line when you press Command-', the entire line is commented out. If you only want to comment out the text to the right of the insertion point, press, ' or use // or Rem.

Example The following code uses comments to document the results of Boolean comparisons:

```
Dim a As Boolean
Dim b As Boolean
Dim c As Boolean
Dim d As Boolean
a=True
b=False
d=a Or b ' Evaluates to True
d=b And c ' Evaluates to False
```

See Also Rem statement; // operator.

* Operator

Used to multiply two numbers.

Syntax

result=expression1 * expression2

Part	Type	Description
result	Number	The product of *expression1* and *expression2*.
expression1	Number	Any numeric expression.
expression2	Number	Any numeric expression.

Examples

This example stores the product of two numbers in a variable:

```
Dim x as Integer
x=50*30   //x is 1500
```

+ Operator

Used to sum two numbers or concatenate two string values.

Syntax

The **+** operator has two syntaxes:

result=expression1 + expression2

Part	Type	Description
result	Number	The sum of *expression1* and *expression2*.
expression1	Number	Any numeric expression.
expression2	Number	Any numeric expression.

result=expression1 + expression2

Part	Type	Description
result	String	The concatenation of *expression1* and *expression2*.
expression1	String	Any string expression.
expression2	String	Any string expression.

Examples

This example adds several numbers together:

```
Dim x as Integer
x=1+2+3
```

This example concatenates two strings and stores the result in a variable:

```
Dim name as String
name="REAL"+"basic" //name = REALbasic
```

See Also Str function.

/ Operator

Used to perform floating point division between two numbers.

Syntax ***result=expression1 / expression2***

Part	Type	Description
result	Number	The division of *expression1* and *expression2*.
expression1	Number	Any numeric expression.
expression2	Number	Any numeric expression.

Notes Use this operator when you require division that considers the decimal portion of the expressions.

Examples This example stores the quotient of two numbers in a variable:

```
Dim x as Double
x=50.56/30.34   //x is 1.6664469
```

See Also \, Mod operators.

// Operator

Used to add comments to code.

Syntax **// text string**

Notes REALbasic's compiler will ignore any comments you enter following the // operator. Comments are not included in stand-alone applications.

The ' (apostrophe) or Rem statement can be used to comment code as well. Comments can appear on separate lines or on the same line as executable code, provided they are to the right of any code that will execute. Valid comments appear in the Code Editor in red.

The Command-' keyboard shortcut toggles the selected lines of code between commented out and 'live' states.

Example

The following code uses comments to document the results of Boolean comparisons:

```
Dim a As Boolean
Dim b As Boolean
Dim c As Boolean
Dim d As Boolean
a=True
b=False
d=a Or b // Evaluates to True
d=b And c // Evaluates to False
```

See Also

Rem statement; ' operator.

< Operator

Used to determine whether one quantitative or alphabetic expression is smaller than another. String comparisons are case-insensitive.

Syntax

result=expression1 < expression2

Part	Type	Description
result	Boolean	Returns True if *expression1* is less than *expression2*.
expression1	String, Number or date	Any alphabetic or quantitative expression.
expression2	String, Number or date	Any alphabetic or quantitative expression.

Note

Use StrComp to do a case-sensitive string comparison.

Example

This example uses the < operator to evaluate a and b.

```
Dim a,b as Integer
If Editfield1.text <> "" and EditField2.text <> "" then
a=Val(Editfield1.text)
b=Val(EditField2.text)
  If a<b then
    MsgBox "A is Less Than B!"
  else
    Beep
  end if
  else
    MsgBox "Please enter values into both boxes!"
end if
```

See Also

>, >=, <=, =, <>, and StrComp operators.

<= Operator

Used to determine whether one alphabetic or quantitative expression is smaller than or equal to another. String comparisons are case-insensitive.

Syntax

result=expression1 <= expression2

Part	Type	Description
result	Boolean	Returns True if *expression1* is less than or equal to *expression2*.
expression1	String, Number, or Date	Any alphabetic or quantitative expression.
expression2	String, Number, or Date	Any alphabetic or quantitative expression.

Note

Use StrComp to do a case-sensitive string comparison.

Examples

This example uses the <= operator to evaluate a and b.

```
Dim a,b as Integer
If Editfield1.text <> "" and EditField2.text <> "" then
a=Val(Editfield1.text)
b=Val(EditField2.text)
  If a<=b then
    MsgBox "A is less than or equal to B!"
  else
    Beep
  end if
  else
    MsgBox "Please enter values into both boxes!"
end if
```

The following example tests whether a FolderItem is Nil before assigning it to the Movie property of a MoviePlayer control.

```
Dim f As FolderItem
f=GetOpenFolderItem("video/quicktime")
If f<>Nil then
  MoviePlayer1.movie=f.OpenAsMovie
End if
```

See Also

>, >=, <, =, <>, and StrComp operators.

<> Operator

Used to determine whether one expression is not equal to another. String comparisons are case-insensitive.

Syntax

result=expression1 <> expression2

Part	Type	Description
result	Boolean	Returns True if *expression1* is not equal to *expression2*.
expression1	String, Number, Object, or Date	An object or alphabetic or quantitative expression.
expression2	String, Number, Object, or Date	An object or alphabetic or quantitative expression.

Notes

The data types of *expression1* and *expression2* must match. The **<>** operator is also used with objects to test whether they are Nil. For example, when you use the NewMemoryBlock function to create a new object, test whether the new object is Nil before proceeding.

Use StrComp to do a case-sensitive string comparison.

Example

The following example uses <> to test for blank EditFields.

```
Dim a,b as Integer
If Editfield1.text <> "" and EditField2.text <> "" then
a=Val(Editfield1.text)
b=Val(EditField2.text)
  If a<b then
    MsgBox "A is Less Than B!"
  else
    Beep
  end if
  else
    MsgBox "Please enter values into both boxes!"
end if
```

See Also

>, >=, <, <=, =, and StrComp operators.

= Operator

Used to determine whether one expression is equal to another. String comparisons are case-insensitive.

Syntax

result=expression1 = expression2

Part	Type	Description
result	Boolean	Returns True if *expression1* is equal to *expression2*.
expression1	String, Number, Boolean, or Color	Any alphabetic or quantitative expression.
expression2	String, Number, Boolean, or Color	Any alphabetic or quantitative expression.

Notes

The data types of *expression1* and *expression2* must match.

Use StrComp to do a case-sensitive string comparison.

Example

The following example tests whether two strings are equal.

```
Dim a,b as String
If Editfield1.text <> "" and EditField2.text <> "" then
a=Editfield1.text
b=EditField2.text
  If a=b then
    MsgBox "A is equal to B!"
  else
    Beep
  end if
  else
    MsgBox "Please enter values into both boxes!"
end if
```

See Also

>, >=, <, <=, <>, and StrComp operators.

> Operator

Used to determine whether one alphabetic or quantitative expression is larger than another. String comparisons are case-insensitive.

Syntax

result=expression1 > expression2

Part	Type	Description
result	Boolean	Returns True if *expression1* is larger than *expression2*.
expression1	String, Number or Date	Any alphabetic or quantitative expression.
expression2	String, Number or Date	Any alphabetic or quantitative expression.

Notes

The data types of *expression1* and *expression2* must match.

Use StrComp to do a case-sensitive string comparison.

Example

The following example tests whether one number is larger than another.

```
Dim a,b as Integer
If Editfield1.text <> "" and EditField2.text <> "" then
a=Val(Editfield1.text)
b=Val(EditField2.text)
  If a>b then
    MsgBox "A is Greater than B!"
  else
    Beep
  end if
  else
    MsgBox "Please enter values into both boxes!"
end if
```

See Also

>=, <, <=, =, <>, and StrComp operators.

>= Operator

Used to determine whether one alphabetic or quantitative expression is greater than or equal to another. String comparisons are case-insensitive.

Syntax

result=expression1 >= expression2

Part	Type	Description
result	Boolean	Returns True if *expression1* is larger than or equal to *expression2*.
expression1	String, Number or Date	Any alphabetic or quantitative expression.
expression2	String, Number or Date	Any alphabetic or quantitative expression.

Notes

The data types of *expression1* and *expression2* must match.

Use StrComp to do a case-sensitive string comparison.

Example

The following example tests whether one number is greater than or equal to another.

```
Dim a,b as Integer
If Editfield1.text <> "" and EditField2.text <> "" then
a=Val(Editfield1.text)
b=Val(EditField2.text)
  If a>=b then
    MsgBox "A is greater than or equal to B!"
  else
    Beep
  end if
  else
    MsgBox "Please enter values into both boxes!"
end if
```

See Also

>, <=, <, =, <>, and StrComp operators.

\ Operator

Used to perform Integer division between two numbers.

Syntax

result=expression1 \ expression2

Part	Type	Description
result	Number	The Integer division of *expression1* and *expression2*.
expression1	Number	Any numeric expression.
expression2	Number	Any numeric expression.

Notes

Use this operator when you require division that *does not* consider the decimal portion of the expressions.

Examples

This example stores the result of a division of two numbers in a variable:

```
Dim x as Integer
x=50.56\30.34   //x is 1
```

See Also

/, Mod operators.

Abs Function

Returns the absolute value of the number specified.

Syntax

***result*=Abs(*value*)**

Part	Type	Description
value	Double	The number you wish the absolute value of.
result	Double	The absolute value of *value*.

Notes

The **Abs** function returns the positive equivalent of the value specified.

Examples

These examples use the **Abs** function to return the absolute values of the numbers specified.

```
Dim d as Double
d=Abs(23.9)  //returns 23.9
d=Abs(-23.9) //returns 23.9
```

See Also

Sign function.

Acos Function

Returns the arccosine of the value specified. The arccosine is the angle whose cosine is *value*. The returned angle is given in radians.

Syntax

***result*=Acos(*value*)**

Part	Type	Description
result	Double	The arc cosine of *value*.
value	Double	The value you want the arc cosine of.

Notes

The **Acos** function returns the angle (in radians) of the cosine passed to it. To convert the result from radians to degrees, multiply it by 180/PI.

Examples

This example uses the **Acos** function to return the arc cosine of a number.

```
Dim d as Double
Const PI=3.14159265358979323846264338327950
d=Acos(.5) //returns 1.0471976
d=Acos(.5)*180/PI //returns 60
```

See Also

Cos function.

ADSP4DServer Class

Used to open a 4th Dimension/4D Server data source via ADSP. Please use this class rather than Open4DDatabaseByADSP.

Super Class

Database

Properties

Name	Type	Description
Type	String	Description of the connection.

Notes

In order to use this class, you must install in your plugins folder the database plugin specific to the version of 4D that you want to access. The version number is appended to the classname, e.g., "ADSP4DServer60". If you use a 4D data source on Windows, you should copy the 4DOpen.DLL file (provided by 4D) into the same directory as your application. This will ensure that the application can find this library and that the proper version of the library is used.

The set of database plug-ins is included on the REALbasic CD, but you may find more recent versions at the REAL Software web site, www.realsoftware.com.

See Also

TCPIP4DServer, Database classes; Open4DDatabaseByADSP, Open4DDatabaseByTCPIP functions.

And Operator

Used to perform a logical comparison of two boolean expressions.

Syntax

result=expression1* And *expression2

Part	Description
result	A boolean value.
expression1	Any valid boolean expression.
expression2	Any valid boolean expression.

Notes

If both expressions evaluate to True, then *result* is True. If either expression evaluates to False then *result* is False.

Example

This example uses the **And** operator to perform logical comparisons.

```
Dim a As Boolean
Dim b As Boolean
Dim c As Boolean
Dim d As Boolean
a=True
b=True
d=a And b //Returns True
d=a And c //Returns False
```

See Also

Not, Or operators.

App Object

An intrinsic instance of the Application class that represents the application.

Syntax

App.*property=value*

App.*method*

Notes

The **App** object represents the application itself (as opposed to a window or control). This allows you access to the Application object's properties, events, and methods without having to explicitly store a reference to it. If you have created a class that is a subclass of type Application, the **App** object will return a reference to that class.

Examples

This example changes the application's MouseCursor to the WatchCursor:

```
App.MouseCursor=WatchCursor
```

Note: Access to the application's resource fork is supported only on Macintosh. Check the value of the TargetMacOS constant before attempting to open a resourcefork.

The following example assigns values to the (user-defined) Separator property of the **App** class in its Open event handler. It will be used to specify full pathnames that are correct on both Macintosh and Windows.

```
#If TargetMacOS
  Separator=":"
#Endif
#If TargetWin32
  Separator="\"
#Endif
```

You use the **App** function to access the Separator property in a method belonging to an object other than the app class:

```
Dim f as FolderItem
Dim s as String
s="HardDisk"+App.Separator+"Applications"+App.Separator+"FrameMaker"
f=GetFolderItem(s)
If f <> Nil then
  If f.Exists then
    // use the folderitem....
  Else
    MsgBox "The folderitem does not exist!"
  End if
Else
  MsgBox "The path "+s+" is invalid."
End if
```

See Also Application, MDIWindow classes; System object.

Append Method

Appends a new element to the end of an array, increasing the size of the array by one.

Syntax ***array*.Append *value***

Part	Description
array	Required. The array to be appended.
value	The value to be assigned to the new array element.

Notes The **Append** method works with one-dimensional arrays only.

Example This example appends a new element to the end of the aNames array.

```
aNames.append "Bill"
```

See Also Dim statement, Insert method, Redim method, Remove method, Ubound function.

AppleEvent Class

AppleEvent objects can be used to communicate with other Macintosh applications and the Macintosh System software. If you are compiling your application for use on non-Macintosh operating systems, be sure to check the global Boolean constants,

TargetMacOS and TargetWin32. These constants are set automatically in the built application and return True if the application is running on the respective operating system.

Super Class

Object

Properties

Name	Type	Description
BooleanParam	ParameterName as String	A Boolean being passed as a parameter in the AppleEvent.
DescListParam	ParameterName as String	An AppleEventDescList object being passed as a parameter in the AppleEvent.
DoubleParam	ParameterName as String	A double being passed as a parameter in the AppleEvent.
EnumeratedParam	ParameterName as String	A four character String being passed as a parameter in the AppleEvent.
FolderItemParam	ParameterName as String	A FolderItem being passed as a parameter in the AppleEvent.
IntegerParam	ParameterName as String	An Integer (passed as a String) being passed as a parameter in the AppleEvent.
MacTypeParam	ParameterName as String	A four character String being passed as a parameter in the AppleEvent.
ObjectSpecifierParam	ParameterName as String	An AppleEventObjectSpecifier being passed as a parameter in the AppleEvent.
Ptr	Integer	Pointer to underlying AppleEvent data structure, for use with Declare statements.
RecordParam	ParameterName as String	An AppleEventRecord being passed as a parameter in the AppleEvent.
ReplyBoolean	Boolean	A reply in Boolean form. True means that the application successfully handled the message.
ReplyDescList	AppleEventDescList	A reply in AppleEventDescriptorList form.
ReplyDouble	Double	A reply in double form.
ReplyEnumerated		
ReplyFolderItem	FolderItem	A reply in FolderItem form.
ReplyInteger	Integer	A reply in integer form.
ReplyMacType	String	
ReplyObjectSpecifier	AppleEventObjectSpecifier	A reply in AppleEventObjectSpecifier form.

Name	Type	Description
ReplyPtr	Integer	A reply in pointer form to the underlying AppleEvent structures for use with Declare statements.
ReplyRecord	AppleEventRecord	A reply in AppleEventRecord form.
ReplySingle	ParameterName as String	A reply in Single form.
ReplyString	String	A reply in String form.
SingleParam	ParameterName as String	A single being passed as a parameter in the AppleEvent.
StringParam	ParameterName as String	A String being passed as a parameter in the AppleEvent.
TimeOut	Integer	Timeout time in seconds -1 = use default 2 = no timeout

Methods

Name	Parameters	Description
LoadFromTemplate	Template as AppleEventTemplate	Loads an AppleEvent template
Send		Returns True if the AppleEvent was successfully sent and False if it was not. True does not mean that the receiving application successfully handled the message. Use the ReplyBoolean property for that.
SetNullParam	ParameterName as String	Sets the parameter specified by the Keyword to a null.

Notes

AppleEvent objects are used to send and receive information between your application and other Macintosh applications or the Mac OS. To send an AppleEvent from your application to another application, create an AppleEvent with the NewAppleEvent function, fill in the AppleEvent's properties with any necessary data, then call the AppleEvent's Send function to send it.

AppleEvents can also be received by your application. When your application receives an AppleEvent, the Application object's HandleAppleEvent event is executed and the AppleEvent is passed to the event as a parameter. For more information on receiving AppleEvents, see the Application class.

Replying To An AppleEvent

When an AppleEvent is received (via an AppleEvent handler) the ReplyBoolean, ReplyInteger, and ReplyString properties can be used to automatically send a reply back to the application that sent the AppleEvent. When sending an AppleEvent (via the Send method) the ReplyBoolean, ReplyInteger, and ReplyString properties can be

used to get any reply the target application has sent back once it receives the AppleEvent.

For example, the line:

```
e.ReplyBoolean=True
```

indicates that the application successfully handled the AppleEvent message.

To determine whether the other application successfully handled a message, use code such as:

```
Dim ae as AppleEvent
Dim s as String
.
.
If ae.Send then //AE successfully sent
 If ae.replyBoolean then
  s = "Yes (handled)"
 else
  s = "Yes (unhandled)"
  end if
 else
 s = "No"
end if
```

For more information on AppleEvents, please refer to Apple's Developer section in the internet.

Examples

This example uses an AppleEvent to tell the Finder to open the Appearance control panel:

```
Dim a as AppleEvent
a = NewAppleEvent("aevt", "odoc", "MACS")
a.FolderItemParam("----")=ControlPanelsFolder.Child("Appearance")
If Not a.Send Then
  MsgBox "The Appearance control panel could not be opened."
End if
```

In this example, the SimpleText application (which must be running for this particular example to work) is instructed to open two documents ("My Document" and "My Other Document") that are located in the folder with the REALbasic project:

```
Dim a as AppleEvent
Dim list as AppleEventDescList
a = NewAppleEvent("aevt", "odoc", "ttxt")
list = New AppleEventDescList
list.AppendFolderItem GetFolderItem("My Document")
list.AppendFolderItem GetFolderItem("My Other Document")
a.DescListParam("----") = list
If Not a.Send Then
  MsgBox "The AppleEvent could not be sent."
End If
```

This example displays the version of the Mac OS that is running on the user's computer:

```
Dim ae as AppleEvent
Dim obj as AppleEventObjectSpecifier
Dim version as String

If TargetMacOS then
 ae = NewAppleEvent("core","getd","MACS")
 obj=GetPropertyObjectDescriptor(nil, "ver2")
 ae.ObjectSpecifierParam("----") = obj

 If Not ae.send() then
  MsgBox "The event could not be sent."
 else
  version = ae.ReplyString
  If (version <> "") then
   MsgBox "The system version is apparently """ + version + """."
  else
   MsgBox "This version of the system software does not support this event."
  end if
 end if
else
 MsgBox "This is not a Macintosh!"
end if
```

See Also

AppleEventDescList Class, AppleEventObjectSpecifier Class, Application Object, NewAppleEvent function, #If…#Else…#Endif statement, TargetMacOS, TargetPPC, Target68K, TargetWin32, DebugBuild constants.

AppleEventDescList Class

Used to send complex information to other applications via AppleEvents.

Super Class

Object

Properties

Name	Type	Description
Count	Integer	The number of items in the list.
FolderItemItem	Index as Integer	A FolderItem array to be passed with the AppleEvent.
IntegerItem	Index as Integer	An Integer array to be passed with the AppleEvent.
RecordItem	Index as Integer	An AppleEventRecord array to be passed with the AppleEvent.
StringItem	Index as Integer	A StringItem array to be passed with the AppleEvent.

Methods

Name	Parameters	Description
AppendBoolean	Value as Boolean	Adds a new Boolean value to the array
AppendDescList	Value as AppleEventDescList	Adds a new AppleEventDescList value to the array.
AppendFolderItem	Value as FolderItem	Adds a new FolderItem to the FolderItemItem array property.
AppendInteger	Value as Integer	Adds a new Integer to the IntegerItem array property.
AppendObjectSpecifier	Value as AppleEventObjectSpecifier	Adds a new AppleEventObjectSpecifier object to the array.
AppendRecord	Value as AppleEventRecord	Adds a new AppleEventRecord to the RecordItem array property.
AppendString	Value as String	Adds a new String to the StringItem array property.
BooleanItem	Index as Integer	Returns Boolean *Index* item.
DescListItem	Index as Integer	Returns AppleEventDescList *Index* item
ObjectSpecifierItem	Index as Integer	Returns AppleEventObjectSpecifier *Index* item

Examples

See the AppleEvent class for an example.

See Also

AppleEvent class, AppleEventRecord class, NewAppleEvent function

AppleEventObjectSpecifier Class

An AppleEventObjectSpecifier specifies an object that an AppleEvent can be performed on.

Super Class Object

Properties None

Events None

Methods None

Notes An AppleEventObjectSpecifier can represent any object such as a document or a pushbutton. AppleEventObjectSpecifers are returned by several functions and are passed to the ObjectSpecifierParam property of an AppleEvent Class object.

Examples See the examples for the AppleEvent Class.

See Also AppleEvent class; GetIndexedObjectDescriptor, GetNamedObjectDescriptor, GetOrdinalObjectDescriptor, GetPropertyObjectDescriptor, GetRangeObjectDescriptor, GetStringComparisonObjectDescriptor, GetTestObjectDescriptor functions.

AppleEventRecord Class

A record that contains a series of different data types. AppleEventRecords can be passed using AppleEvents.

Super Class Object

Properties

Name	Type	Description
BooleanParam	ParameterName as String	A boolean being passed as a parameter in the AppleEventRecord.
DescListParam	ParameterName as String	An AppleEventDescList object being passed as a parameter in the AppleEventRecord.
DoubleParam	ParameterName as String	A Double being passed as a parameter in the AppleEventRecord.

Name	Type	Description
EnumeratedParam	ParameterName as String	A four character String being passed as a parameter in the AppleEventRecord.
FolderItemParam	ParameterName as String	A FolderItem being passed as a parameter in the AppleEventRecord.
IntegerParam	ParameterName as String	An integer (passed as a String) being passed as a parameter in the AppleEventRecord.
MacTypeParam	ParameterName as String	A four character String being passed as a parameter in the AppleEventRecord.
ObjectSpecifierParam	ParameterName as String	An AppleEventObjectSpecifier being passed as a parameter in the AppleEventRecord.
RecordParam	ParameterName as String	An AppleEventRecord being passed as a parameter in the AppleEventRecord.
SingleParam	ParameterName as String	A Single being passed as a parameter in the AppleEventRecord.
StringParam	ParameterName as String	A String being passed as a parameter in the AppleEventRecord.

Methods

Name	Parameters	Description
LoadFromTemplate	Template as AppleEventTemplate	Loads an Apple Event template.
SetNullParam	ParameterName as String	Sets the parameter specified by the Keyword to a null.

See Also

AppleEvent class.

AppleEventTarget Class

Used to send Apple Events to applications on other computers.

Super Class

Object

Properties

Name	Type	Description
Remote	Boolean	True if a different computer.
Computer	String	Name of computer.
PortType	String	Creator code that identifies the application.
Zone	String	AppleTalk zone.

Methods

Name	Parameters	Description
NewAppleEvent	EventClass as String EventID as String	Creates a new Apple Event. Returns an AppleEvent.

AppleEventTemplate Class

A template object for creating AppleEvent objects.

Super Class

Object

Properties

Name	Parameters	Description
IntegerParam	Name as String	An Integer being passed as a parameter in the Apple Event.
StringParam	Name as String	A String being passed as a parameter in the Apple Event.
FolderItemParam	Name as String	A FolderItem being passed as a parameter in the Apple Event.

AppleMenuFolder Function

Used to access the Apple Menu Items folder. On Windows, it accesses the Programs directory in the Start Menu directory.

Syntax

***result*=AppleMenuFolder**

Part	Type	Description
result	FolderItem	A FolderItem that represents the Apple Menu Items folder.

Notes

Use the **AppleMenuFolder** function to access the Apple Menu Items folder on Macintosh or the FolderItem for the Start menu items on Windows.

The **AppleMenuFolder** function provides a way to access the Apple Menu Items folder that will work under different language systems, and will continue to work even if Apple reorganizes the System folder. If your application will be used on Windows, check the value of the TargetWin32 global constant before calling **AppleMenuFolder**.

Examples

This example places all the names of the items in the Apple Menu Items folder in a ListBox.

```
Dim i as Integer
Dim f as FolderItem
f=AppleMenuFolder
For i=1to f.Count
   ListBox1.AddRow f.Item(i).Name
  Next
Exception err as NilObjectException
  MsgBox "There is no Apple Menu Folder!"
```

See Also

ControlPanelsFolder, DesktopFolder, ExtensionsFolder, FontsFolder, PreferencesFolder, ShutDownItemsFolder, StartupItemsFolder, SystemFolder, TemporaryFolder, TrashFolder functions.

Application Class

An instance of the Application Class called "App" is created automatically when your application is opened. It is used to handle menu items when no windows are open or for handling menu items that should respond in the same way regardless of which window is open. It also has events that inform you when the user has double-clicked on one of your application's documents and receives AppleEvents sent to your application as well.

Super Class

Object

Properties

Name	Type	Description
BalloonHelpVisible	Boolean	Supported on Windows only. When set to False, Balloon Help is disabled; True enables Balloon Help.
BugVersion	Integer	Bug version of the application, corresponding to the version information stored via the Build Settings dialog box.

Name	Type	Description
DockItem		(Mac OS X only). Enables you to manipulate the dock item associated with the application. The DockItem property has two methods, UpdateNow and ResetIcon. ResetIcon resets the icon to its original state (default appearance). Use the methods of the Graphics class to modify the appearance of the icon. Since a Mac OS X icon is intended to be scaled automatically, you should design it as a 128x128 pixel icon. Call the UpdateNow method to redraw the icon. You can also use the ClearRect property of the Graphics class to start over from a blank icon. Anything you can do with a Graphics object you are able to do with the Dock's Graphics object (like drawing in a picture or using a Shape 2D). The DockItem property of the Window class enables you to control the dock item for individual windows.
LongVersion	String	Long version of the application, corresponding to the version information stored via the Build Settings dialog box.
MajorVersion	Integer	Major version of the application, corresponding to the version information stored via the Build Settings dialog box.
MenuBar	MenuItem	Represents the application's global menubar. Its children are the menus. You can replace the entire menubar by assigning a new MenuItem to MenuBar. Each Window can have a menubar assigned to it via its MenuBar property. If no menubar is assigned to the window, the application's menubar is used.
MinorVersion	Integer	Minor version of the application, corresponding to the version information stored via the Build Settings dialog box.
MouseCursor	MouseCursor	The cursor that is displayed while the application is running. If the Application class MouseCursor is not Nil, the non-Nil MouseCursor properties belonging to any Windows or Controls are ignored. Names are: WatchCursor, IBeamCursor, and ArrowCursor. MouseCursors may also be assigned from resource files.
NonReleaseVersion	Integer	NonRelease version of the application, corresponding to the version information stored via the Build Settings dialog box.
PackageInfo	String	Package Info of the application, corresponding to the version information stored via the Build Settings dialog box.
RegionCode	Integer	Region Code of the application, corresponding to the version information stored via the Build Settings dialog box. Not supported on Windows.

Name	Type	Description
ResourceFork	ResourceFork	Used to access the application's resources. Supported only on Macintosh. Check the value of TargetMacOS before attempting to access the application's resources.
ShortVersion	String	Short version of the application, corresponding to the version information stored via the Build Settings dialog box.
StageCode	Integer	Stage Code of the application, corresponding to the version information stored via the Build Settings dialog box.

Methods

Name	Parameters	Description
NewDocument		Calls the NewDocument event handler.
OpenDocument	File as FolderItem	Calls the OpenDocument event handler.

Events

Name	Parameters	Description
Activate		The application is being activated.
Close		The application is quitting.
Deactivate		The application is being deactivated.
EnableMenuItems		Executes when the user clicks in the menubar to give you the opportunity to determine which menu items to enable. Also executes when the application opens if no default window is selected in the Project Settings dialog box and when the last window is closed.
HandleAppleEvent	Event as AppleEvent, EventClass as String, EventID as String	An AppleEvent has been received. Return True to accept the AppleEvent and False to reject it.
NewDocument		The user launched the application by double-clicking the application icon or the NewDocument method was called.
Open		The application is opening.
OpenDocument	File as FolderItem	The user has double-clicked on a document whose creator matches the application's creator or the OpenDocument method was called.

Name	Parameters	Description
UnhandledException	error as RuntimeException	Receives otherwise missed exceptions—exceptions that could have been caught in an Exception Block using one of the Runtime exception handlers. If the appropriate Exception Block is missing, the runtime exception error triggers this event and allows you to handle exceptions here. See the example. Returns a Boolean. If it returns True, the standard alert indicating an unhandled exception and quit behavior are suppressed. Return False to allow REALbasic's default unhandled exception error processing to occur. See the RuntimeException class.

Notes

REALbasic adds a class based on the Application class to the Project window when you create a new project. This new subclass will give you access to the **Application** object's methods and events.

The new subclass has its own menu handlers which can be used to handle menus items when no windows are open or for menu items that should call the same menu handler regardless of which window is frontmost. You can change the global menubar by assigning a different menubar to its MenuBar property.

See the Control class for information on changing the cursor and adding cursors to your project.

The App function returns a reference to the **Application** object. See the App function for more information.

Resource files (files of type "rsrc") created with ResEdit or equivalent can be added to your project by dragging each resource file into the Project window. These resources are then accessible through the **Application** object's ResourceFork property. The resources from all your resource files will be copied into the built Macintosh application. In the case of a conflict, later resource files overwrite earlier ones, where the files are written in the order in which they appear in the Project window.

The application resources are read-only. If you need to write to resources you will need to use an external resource file. See the ResourceFork class for more information.

The **Application** class properties that set version information work in the REALbasic IDE and in Macintosh builds only. Version information is not available on Windows builds.

Note: Access to the resourcefork is supported only on Macintosh. Check the value of the TargetMacOS constant before attempting to access the application's resourcefork.

Example

The following code in the Action event of a PushButton causes an OutofBoundsException runtime error when the counter, i, reaches the value of FontCount.

```
Dim i as Integer
For i=1 to FontCount
 ListBox1.Addrow Font(i)
Next
```

Since there is no exception handler within the method, the runtime exception is passed up to the Application class, triggering the UnhandledException event.

This code in the UnhandledException event of the App class "catches" the unhandled OutOfBoundsException. Of course, it catches all unhandled OutOfBoundsExceptions throughout the application, so it doesn't know where the error occurred. You could instead place an Exception Block within the PushButton's Action event so that you can provide more specific diagnostics.

```
Function UnhandledException(error as RuntimeException) as Boolean
 If error IsA OutOfBoundsException then
  MsgBox "An OutOfBounds Exception error has occurred!"
 End if
 Return True
End Function
```

See Also

App, System, ResourceFork objects; AppleEvent, MDIWindow classes.

ApplicationSupportFolder Function

Returns a FolderItem that references the Application Support folder in the active System folder. On Windows, it references the c:\Documents and Settings*username*\Local Settings\Application Data\ directory, if available. If not, it references the Windows directory.

Syntax

***result*=ApplicationSupportFolder**

Part	Type	Description
result	FolderItem	A FolderItem that refers to the Application Support folder.

Notes

The Application Support folder may contain code and data files needed by third-party applications. These files should usually not be written to after they are installed. In general, files deleted from this folder remove functionality from an application, unlike files in the Preferences folder, which should be non-essential.

The Application Support folder is supported by Mac OS 8 and later.

Example

The following example displays the full path to the Application Support folder, if there is one.

```
Dim f as FolderItem
f=ApplicationSupportFolder
If f <> Nil then
  MsgBox f.absolutePath
else
  MsgBox "No Application Support folder found."
end if
```

See Also

DesktopFolder, ControlPanelsFolder, ExtensionsFolder, FontsFolder, PreferencesFolder, ShutDownItemsFolder, StartupItemsFolder, SystemFolder, TemporaryFolder, TrashFolder functions.

ArcShape Class

Used for drawing arcs in a vector graphics environment.

Super Class

OvalShape

Properties

Name	Type	Description
ArcAngle	Double	The angle of the arc, in radians.
StartAngle	Double	The starting angle, in radians.

Notes

A StartAngle of 0 means due east, i.e., or a positive X value with Y=0. Positive angles are clockwise, just as with Object2D.rotation. The value of ArcAngle can be positive or negative, extending from the starting angle. If the angle is filled, it produces a wedge; this may be useful for making pie charts, for example.

Examples

The following method, when called from the Open event of a Canvas control, creates an arc. The Group2D object, d, is declared as a property of the window.

```
Dim a as ArcShape
Dim p as Picture
a=New ArcShape
d=New Group2D
p=New Picture(100,50,0) //depth of zero for vector graphics only
a.arcAngle=1.57
a.startAngle=-1.57
a.fillcolor=RGB(255,0,127)
d.append(a)
Canvas1.Backdrop=p
Canvas1.Backdrop.Objects=d
```

The following line in the Canvas's Paint event draws the arc:

```
g.drawobject d,100,80
```

This arc looks like this:

See Also

CurveShape, FigureShape, FolderItem, Group2D, Graphics, Object2D, OvalShape, Picture, PixmapShape, RectShape, RoundRectShape, StringShape classes.

Array Bounds must be Integers Error

You used a non-integer in a Dim statement when trying to declare an array.

Examples

```
Dim aPicture (5.2) as String
Dim aProps (True) as Boolean
```

Asc Function

Returns as an Integer, the ASCII value for the first character of a String.

Syntax

***result*=Asc(*string*)**

OR

***result=stringVariable*.Asc**

Part	Type	Description
result	Integer	The ASCII value of the first character of *string*.
string	String	Any valid string expression.
StringVariable	String	Any variable of type String.

Notes

The **Asc** function returns the ASCII code for the first character in the String passed. Characters 0 through 127 are the standard ASCII set. They will be the same on practically every platform. Characters 128 through 255 are the extended ASCII set and will be different depending on the platform where the data originated. This is important to keep in mind when using the **Asc** function with data from files that have come from a user running a different operating system.

If you need to get the ASCII code of the first byte of the string rather than the first character, use the AscB function.

Examples

This example uses the **Asc** function to get the ASCII value of a character.

```
Dim a as Integer
a = Asc("@") //returns 64
```

This example gets the ASCII value of a bullet character:

```
Dim s as String
Dim n as Integer
s="•"
n=s.Asc
```

See Also

Chr, InStr, Left, Len, Mid, Right functions.

AscB Function

Returns as an Integer, the ASCII value for the first byte of a String.

Syntax

***result*=AscB(*string*)**

OR

***result*=*stringVariable*.AscB**

Part	Type	Description
result	Integer	The ASCII value of the first character of *string*.
string	String	Any valid string expression.
StringVariable	String	Any variable of type String.

Notes

The **AscB** function returns the ASCII code for the first byte in the String passed. Characters 0 through 127 are the standard ASCII set. They will be the same on practically every platform. Characters 128 through 255 are the extended ASCII set and will be different depending on the platform where the data originated. This is important to keep in mind when using the **AscB** function with data from files a user running a different operating system.

If you need to get the ASCII code of the first character of the string rather than the first byte, use the Asc function.

AscB should be used instead of Asc when the string represents binary data or when your application will run on a one-byte character set (such as the US system) and you want case-sensitivity.

Examples

This example uses the **AscB** to get the ASCII value of the first byte of the string passed.

```
MsgBox Str(AscB("a")) //returns 97
MsgBox Str(AscB("A")) //returns 65
```

See Also

Asc, ChrB, InStrB, LeftB, LenB, MidB, RightB functions.

Asin Function

Returns the arcsine of the value specified.

Syntax

***result*=Asin(*value*)**

Part	Type	Description
result	Double	The arc sine of *value*.
value	Double	The value you want the arc sine of. *Value* is the Sin of the angle you want and must be between -1 and 1.

Notes

The arcsine is the angle whose sine is *value*. The **Asin** function returns the angle (in radians) of the sine passed to it. To express the arcsine in degrees, multiply the result by 180/PI.

Examples

This example uses the **Asin** function to return the arcsine of a number.

```
Dim d as Double
Const PI=3.14159265358979323846264338327950
d=Asin(.5)  //returns 0.5235988
d=Asin(.5)*180/PI  //returns 30
```

See Also

Sin function.

Assigns Keyword

Used to indicate that a parameter will be passed to a method via the Assignment operator.

Syntax

Assigns *parameter* As *DataType*

Part	Type	Description
parameter		The name of the parameter being declared.
DataType	Any valid data type	The data type of the parameter.

Notes

Use the **Assigns** keyword when you want to assign a value to a parameter of a method with the equals sign rather than as an argument.

The **Assigns** keyword can appear in a Method declaration dialog box for the only parameter passed to the method or, if more than one parameter is being passed, for the last parameter. When **Assigns** is used, you use a different syntax when calling the method. The value for the parameter must be passed using the assignment operator rather than as an argument.

Example

The following declaration uses **Assigns** for the last parameter:

```
Sub theVolume(a as Integer, b as Integer, Assigns c as Integer)
```

When this method is called, only the values for a and b are passed as arguments; the value for c is passed using the assignment operator. For example:

```
theVolume(5,4)=10
```

If you try to use the syntax:

```
theVolume(5,4,10)
```

you will receive a Syntax error.

See Also Assigns can only be used on the last parameter Error.

Assigns can only be used on the last parameter Error

You used the Assigns keyword in a Method declaration for a parameter other than the last parameter. If there is only one parameter, the use of Assigns is permitted.

Example The following declaration uses the Assigns keyword for the first parameter.

```
Sub SquareIt(Assigns x as Double, y as Double)
```

See Also Assigns keyword.

Atan Function

Returns the arctangent of the value specified. The arctangent is the angle whose tangent is *value*.

Syntax ***result*=Atan(*value*)**

Part	Type	Description
result	Double	The arc tangent of *value*.
value	Double	The value you want the arc tangent of.

Notes The **Atan** function returns the angle (in radians) of the number passed to it. To express the arctangent in degrees, multiply the result by 180/PI.

Examples This example uses the **Atan** function to return the arc tangent of a number.

```
Dim d as Double
Const PI=3.14159265358979323846264338327950
d=Atan(1) //returns 0.785398 (PI/4 radians)
d=Atan(1)*180/PI // returns 45
```

See Also Tan, Atan2 functions.

Atan2 Function

Returns the arctangent of the point whose coordinates are *x* and *y*. The arctangent is the angle from the x-axis to a line drawn through the origin (0,0) and a point with coordinates *x, y*.

Syntax

result = **Atan2** (y, x)

Part	Type	Description
result	Double	Arctangent of the point (y, x) in radians.
y	Double	y coordinate of the point.
x	Double	x coordinate of the point.

Notes

The result is expressed in radians. To convert it to degrees, multiply it by 180/PI.

The converse operations are done with Cos and Sin. That is, if you have an angle and want to find an x, y pair along the line described by 0,0 and x,y, you can do so with:

```
x = Cos(angle)*radius
y= Sin(angle)*radius
```

Examples

This example uses the **Atan2** function to return the arctangent of a point directly above the origin.

```
Dim d as Double
Const PI=3.14159265358979323846264338327950
d=Atan2(1,0) //returns 1.57
d=Atan2(1,0)*180/PI //returns 90
```

See Also

Tan, Atan functions.

Beep Method

Plays your computer's system beep sound.

Syntax

Beep

Notes

The **Beep** method plays the Alert sound selected in your Sound or Monitors and Sound Control panel.

Example

This example plays the alert sound three times.

```
Beep
Beep
Beep
```

BevelButton Control 

Used for creating a bevel button. A **BevelButton** can use text, a graphic, a pop-up menu, or several of these interface elements in combination.

Super Class

RectControl

Because this is a RectControl, see the RectControl for other properties and events that are common to all RectControl objects.

Properties

Name	Type	Description
Bevel	Integer	0 - Small bevel 1 - Normal bevel 2 - Large bevel
Bold	Boolean	Applies the bold style to the button caption.
ButtonType	Integer	0 - Button. Remains in 'down' position until mouse is released. 1 - Toggles. Remains in 'down' position until clicked again. 2 - Sticky. Remains in 'down' position when clicked.
Caption	String	The button's text.
CaptionAlign	Integer	Alignment of caption. 0 - Flush left 1 - Flush right 2 - Sys direction 3 - Center
CaptionDelta	Integer	Distance in pixels of the caption from the left of the button.
CaptionPlacement	Integer	Placement of caption relative to graphic (Icon). 0 - Sys Direction 1 - Normally 2 - Right of graphic 3 - Left of graphic 4 - Below graphic 5 - Above graphic
Icon		Name of graphic to use as icon. Drag the graphic to the project window or import it using File ▶ Import. Must be a PICT image. See first example.

Name	Type	Description
IconAlign	Integer	Alignment of graphic within BevelButton 0 - Sys Direction 1 - Center 2 - Left 3 - Right 4 - Top 5 - Bottom 6 - Top left 7 - Bottom left 8 - Top right 9 - Bottom right
IconDx	Integer	Distance in pixels from 'flush' left or right, depending on alignment. If center is chosen, IconDx does nothing.
IconDy	Integer	Distance in pixels from 'flush' top or bottom, depending on alignment. If center is chosen, IconDy does nothing.
Italic	Boolean	Applies the italic style to the button caption.
LockBottom	Boolean	Determines whether the bottom edge of the control should stay at a set distance from the bottom edge of the owning window.
LockLeft	Boolean	Determines whether the left edge of the control should stay at a set distance from the left edge of the owning window. *LockLeft* has no effect unless *LockRight* is True.
LockRight	Boolean	Determines whether the right edge of the control should stay at a set distance from the right edge of the owning window.
LockTop	Boolean	Determines whether the top edge of the control should stay at a set distance from the top edge of the owning window. *LockTop* has no effect unless *LockBottom* is True.
HasMenu	Integer	Controls whether BevelButton control has a popup menu. 0 - No menu 1 - Normal menu 2 - Menu on right
MenuValue	Integer	The number of the menu item the user selects. The first menu item is number zero. A separator cannot be selected, but "counts" as a menu value.
TextFont	String	Name of the font used to display the button caption.
TextSize	Integer	Size of the font used to display the button caption.
Underline	Boolean	Applies the underline style to the button caption.
Value	Boolean	If True, the button initially appears as if it is pressed.

Methods

Name	Parameters	Description
AddRow	Text as String	Adds a row to the bottom of the menu and uses *Text* as the menu item.

Name	Parameters	Description
AddSeparator		Adds a row to the bottom of the menu and uses a separator as the menu "item."
DeleteAllRows		Deletes all rows of the current Bevel button menu.
InsertRow	Row as Integer Text as String	Inserts a row in the position indicated by *Row* using *Text* as the menu item.
List	Row as Integer	Returns the text of the menu item as a string. The first menu item is zero.
RemoveRow	Row as Integer	Removes *Row* from the menu. The first menu item is zero.

Notes

A **BevelButton** can display a graphic, a label (caption), or both.

The first row in a menu has an index of zero.

If the Caption property contains an ampersand character, it will display only if it is preceded by another ampersand character. For example, if you set the Caption property to "Bob & Ray", only "Bob Ray" will display. You need to set it to "Bob && Ray". The ampersand character is used on Windows for keyboard accelerators.

This is done to make Macintosh and Windows applications behave consistently.

Examples

The following example places an icon in the bevel button, aligns it, and specifies the placement of the caption. The graphic, "databasequery," has been dragged into the Project window.

```
Me.icon=databasequery
Me.iconalign=6
Me.captionalign=0
Me.captionplacement=2
Me.icondx=2
Me.icondy=1
```

The following example creates a bevel button menu.

```
Dim s as String
Dim i as Integer
Dim last as Integer
s="Macintosh,Windows,Unix,-,Other"
last=CountFields(s,",")
Me.caption="Platform"
Me.captionalign=0 //flush left
Me.hasMenu=2      //menu on right
For i=1 to last
 me.addRow NthField(s,",",i)
Next
```

When the user selects the "Other" menu item, MenuValue is set to 4 because the separator counts as a menu item even though it cannot be selected.

The following line of code in the **BevelButton's** Action event handler sets the **BevelButton** caption to the value that the user selects:

```
Me.caption=Me.list(Me.MenuValue)
```

See Also PushButton, PopupMenu, ContextualMenu controls.

Bin Function

Returns the binary version of the number passed as a string.

Syntax ***result*=Bin(*value*)**

Part	Type	Description
result	String	*Value* converted to binary.
value	Integer	The number to be converted to binary.

Notes If the value is not a whole number, the decimal value will be truncated.

You can specify binary, hex, or octal numbers by preceding the number with the & operator and the letter that indicates the number base. The letter b indicates binary, h indicates hex, and o indicates octal.

Examples Below are examples of various numbers converted to binary:

```
Dim binVersion As String
binVersion=Bin(15) //returns "1111"
binVersion=Bin(15.5) //returns "1111"
binVersion=Bin(75) //returns "1001011"
binVersion=Bin(&hF) //returns "1111"
```

See Also & operator, Oct and Hex functions.

BinaryStream Class

BinaryStream Class objects are used to read and write data to and from a binary file. The benefit of using BinaryStreams is that you can read from and write to any position in the file. Text files must be read sequentially from the start to the end.

Super Class Object

Properties

Name	Type	Description
EOF	Boolean	The stream has reached the end of the file.
Length	Integer	The length of the file's data fork.
LittleEndian	Boolean	If True, the byte order is assumed to be low byte, high byte.
Position	Integer	The current file position in the binarystream. This property is automatically incremented by all of the Read and Write methods.

Methods

Name	Parameters	Description
Close		Closes the stream (and hence the file opened by the stream).
Read	Count as Integer, [Encoding as TextEncoding]	Reads *Count* characters from the stream starting at the Position property and returns them as a String. The optional parameter *Encoding* enables you to specify the encoding to be defined on the returned string. If you omit this parameter (or pass Nil), then the encoding of the returned string will be Nil.
ReadBoolean		Reads a boolean value form the stream and returns it as a Boolean.
ReadByte		Reads a byte from the stream and returns it as an Integer.
ReadDouble		Reads a double word from the stream and returns it as a Double.
ReadLong		Reads a long word (4 bytes) from the stream and returns it as an Integer.
ReadSingle		Reads a word from the stream and returns it as a Double.
ReadPString	[Encoding as TextEncoding]	Reads a Pascal string from the stream and returns it as a String. The optional parameter *Encoding* enables you to specify the encoding to be defined on the returned string. If you omit this parameter (or pass Nil), then the encoding of the returned string will be Nil.
ReadShort		Reads a short word (2 bytes) from the stream and returns it as an Integer.
Write	Data as String	Writes *Data* to the file starting at the Position property.
WriteBoolean	Value as Boolean	Writes the value passed as a boolean to the stream.
WriteByte	Value as Integer	Writes the value passed as a byte to the stream.
WriteDouble	Value as Double	Writes the value passed as a double word.
WriteLong	Value as Integer	Writes the value passed as a long word (4 bytes) to the stream.

Name	Parameters	Description
WriteSingle	Value as Double	Writes the value passed as a single word.
WritePString	Text as String	Writes the text passed as a Pascal string to the stream.
WriteShort	Value as Integer	Writes the value passed as a short word (2 bytes) to the stream.

Notes

What is the LittleEndian property? Some operating systems store binary values in the reverse order from the Mac OS. If you were using the ReadShort or ReadLong methods to read data from a file that was in little endian format, you would get incorrect data. REALbasic reads data in big endian format. Most Macintosh files are in big endian format. If you are reading a file that is in little endian format, you will need to set this property to true before you begin reading the file. This applies to writing data with WriteShort and WriteLong.

For example, in big endian (like the Mac OS), the value 258 would be stored as:

01 02

while in little endian, it would be stored as:

02 01

If the LittleEndian property is set incorrectly, then you would read the value as 513.

Because REALbasic has the LittleEndian property, you can write your code to be operating system independent. Set the LittleEndian property to True if the file format is intrinsically little endian (i.e. GIF files), otherwise leave it as False.

Example

This example reads each pair of bytes from a file and writes them in reverse order to a new file. The user chooses the source file using the Open-file dialog box and saves the new file using the Save as dialog box.

```
Dim WriteToFile as BinaryStream
Dim ReadFromFile as BinaryStream
Dim f as FolderItem
f=GetOpenFolderItem("text")
If f <> Nil Then
  ReadFromFile=f.OpenAsBinaryFile(False)
  ReadFromFile.littleEndian=True
  f=GetSaveFolderItem("","")
  If f <> Nil Then
    WriteToFile=f.CreateBinaryFile("text")
    While Not ReadFromFile.EOF
      WriteToFile.WriteShort ReadFromFile.ReadShort
    wend
    WriteToFile.close
  End If
  ReadFromFile.close
End If
```

See Also

TextInputStream, TextOutputStream, FolderItem classes.

Bitwise Class

Performs bitwise operations on integers. The methods of the **Bitwise** class replace the BitwiseAnd, BitwiseOr, and BitwiseXor functions.

Super Class

Object

Methods

Name	Parameters	Description
BitAnd	value1 as Integer value2 as Integer	Returns an Integer. Performs a bitwise And on *value1* and *value2*. See the description of BitwiseAnd.
BitOr	value1 as Integer value2 as Integer	Returns an Integer. Performs a bitwise Or on *value1* and *value2*. See the description of BitwiseOr.
BitXor	value1 as Integer value2 as Integer	Returns an Integer. Performs a bitwise Or on *value1* and *value2*. See the description of BitwiseXor.
OnesComplement	value as Integer	Performs a one's compliment operation on *value*.

Name	Parameters	Description
ShiftLeft	value as Integer shift as Integer [,numBits as Integer]	Shifts *value* to the left by *shift*. Only shifts bits within *numBits* field size. *numBits* defaults to 32 bits.
ShiftRight	value as Integer shift as Integer [,numBits as Integer]	Shifts *value* to the right by *shift*. Only shifts bits within *numBits* field size. *numBits* defaults to 32 bits.

Notes

The BitwiseAnd, BitwiseOr, and BitwiseXor functions are superseded by the **Bitwise** class. Please recode to use this class.

Example

```
Dim i as Integer
i=Bitwise.bitand(5,3) //returns 1
i=Bitwise.bitor(5,3) //returns 7
i=Bitwise.bitxor(5,3) //returns 6
```

See Also

BitwiseAnd, BitwiseOr, BitwiseXor functions.

BitwiseAnd Function

Performs a BitwiseAnd on the two values passed and returns the result.

Syntax

***result*=BitwiseAnd(*value1, value2*)**

Part	Type	Description
result	Integer	The BitwiseAnd of *value1* and *value2*.
value1	Integer	A value you wish to perform a bitwiseAnd on.
value2	Integer	Another value you wish to perform a bitwiseAnd on.

Notes

The **BitwiseAnd** function returns an integer that is the result of comparing each bit of the two integers passed and assigning 1 to the bit position in the integer returned if both bits in the same position in the integers passed are 1. Otherwise, 0 is assigned to the bit position.

The Bitwise class supersedes this function. Please perform this operation by calling the Bitwise class's BitAnd method.

Example

Integers are 32 bit numbers. For the purposes of this example, only the last five bits are displayed since the rest would all be zero anyway.

5 in binary is 00101

3 in binary is 00011

The result of **BitwiseAnd**(5, 3) is 1, which is 00001 in binary.

See Also Bitwise class, BitwiseOr, BitwiseXor functions.

BitwiseOr Function

Performs a BitwiseOr on the two values passed and returns the result.

Syntax ***result*=BitwiseOr(*value1, value2*)**

Part	Type	Description
result	Integer	The bitwiseOr of *value1* and *value2*.
value1	Integer	A value you wish to perform a bitwiseOr on.
value2	Integer	Another value you wish to perform a bitwiseOr on.

Notes The **BitwiseOr** function returns an Integer that is the result of comparing each bit of the two integers passed and assigning 1 to the bit position in the integer returned if either of the bits in the same position in the integers passed are 1. Otherwise, 0 is assigned to the bit position.

The Bitwise class supersedes this function. Please perform this operation by calling the Bitwise class's BitOr method.

Example Integers are 32 bit numbers. For the purposes of this example, only the last five bits are displayed since the rest would all be zero anyway.

5 in binary is 00101

3 in binary is 00011

The result of **BitwiseOr**(5, 3) is 7 which is 00111 in binary.

See Also Bitwise class, BitwiseAnd, BitwiseXor functions.

BitwiseXor Function

Performs a BitwiseXor on the two values passed and returns the result.

Syntax

***result*=BitwiseXor(*value1*, *value2*)**

Part	Type	Description
result	Integer	The bitwiseXor of *value1* and *value2*.
value1	Integer	A value you wish to perform a bitwiseXor on.
value2	Integer	Another value you wish to perform a bitwiseXor on.

Notes

The **BitwiseXor** function returns an Integer that is the result of comparing each bit of the two integers passed and assigning 1 to the bit position in the integer returned if both bits in the same position in the integers passed are not equal. Otherwise, 0 is assigned to the bit position.

The Bitwise class supersedes this function. Please perform this operation by calling the Bitwise class's BitXor method.

Example

Integers are 32 bit numbers. For the purposes of this example, only the last five bits are displayed since the rest would all be zero anyway.

5 in binary is 00101

3 in binary is 00011

The result of **BitwiseXor**(5, 3) is 6 which is 00110 in binary.

See Also

Bitwise class, BitwiseAnd, BitwiseOr functions.

Boolean Data Type

Boolean is an intrinsic data type in REALbasic. A **Boolean** can only take on the values of True or False.

Boolean values are False by default but can be set to True using REALbasic's True function and back to False using the False function. Some of the properties of objects in REALbasic are **Boolean** values. In the Properties Window, they appear as checkboxes. For example, most of the controls have an Enabled property that is **Boolean**.

The VarType function returns 11 when passed a **Boolean** variable.

Example

The following statement disables an EditField, making it non-enterable and non-selectable (i.e., the user can't tab into it).

```
EditField1.Enabled=False
```

See Also

True, False, VarType functions.

Bounds3D Class

Represents a bounding volume, useful for simple collision and visibility testing in RB3DSpace controls. A **Bounds3D** can represent either an axis-aligned bounding box or and a bounding sphere.

Super Class

Object

Properties

Name	Type	Description
Center	Vector3D	The center of the sphere or box.
Maximum	Vector3D	The maximum x, y, and z or the bounding box.
Minimum	Vector3D	The minimum x, y, and z of the bounding box.
Radius	Double	The radius of the bounding sphere.
Type	Integer	The type of the bounding object. 1=Sphere 2=Box 0 =Invalid.

Methods

Name	Parameters	Description
Intersects	Other as **Bounds3D**	Returns a Boolean. Tests whether this bounding volume overlaps another. When in doubt, it returns True.
InView	Rb3D as RB3DSpace	Returns a Boolean. Tests whether the Bounds3D volume is at least partly in the viewable part of the 3D world, *RB3D*. When in doubt, it returns True.
ContainsPoint	Pt as Vector3D	Returns a Boolean. Tests whether this bounding volume contains the 3D point, *Pt*.
LineIntersection	Pt1 as Vector3D Pt2 as Vector3D	Returns a Vector3D. Finds the point at which a line through *Pt1* and *Pt2* intersects the bounding volume. If it does not intersect, it returns Nil.
LineSegmentIntersection	Pt1 as Vector3D Pt2 as Vector3D	Returns a Vector3D. Finds the point at which a line through *Pt1* and *Pt2* intersects the bounding volume. If the intersection point is not between *Pt1* and *Pt2*, it returns Nil.

Constructor Syntax

This table shows how to create either a bounding box or sphere.

Syntax	Parameters	Description
New Bounds3D	Center as Vector3D Radius as Double	The constructor for a **Bounds3D** sphere.
New Bounds3D	Minimum as Vector3D Maximum as Vector3D	The constructor for a **Bounds3D** box.

Notes

As indicated in the Methods table, some of the tests are not exact; in hard-to-decide edge cases, **Bounds3D** returns True, which indicates that two objects overlap or an object is visible (InView). This means that a return value of True may indicate a need for more detailed tests, but a return value of False indicates that no further testing is necessary.

Currently, the Line intersection tests treat all **Bounds3D** volumes as bounding spheres.

See Also

RB3DSpace, Vector3D classes.

ByRef Keyword

Used to pass a parameter by reference.

Syntax

ByRef *parameter*

Part	Type	Description
parameter	Any data type	Parameter to be passed by reference.

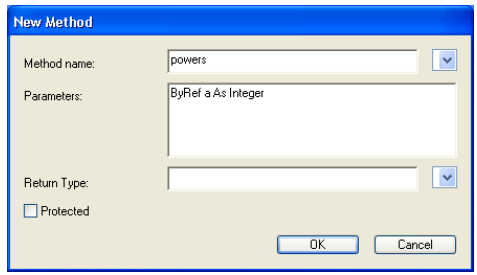
Notes

By default, you pass parameters to methods by value. When you do so, the method receives a copy of the data in the object that you pass. Your method receives the data and can perform operations on it.

If you precede the parameter by the keyword **ByRef**, you pass information by reference. When you pass information by reference, you actually pass a pointer to the object containing the information. The practical advantage of this technique is that the method can *change the values* of each parameter. When you pass parameters by value, you can't do this because the parameter only represents a copy of the data itself.

Example

The following method declaration uses **ByRef**.



The Powers method takes one parameter, an integer, a, that is called Byref.

The method is simply:

```
a=a*a
```

Powers is called in the following code:

```
Dim a as Integer
a=3
powers a
EditField1.text=Str(a)
```

The EditField displays the value of 9.

See Also

ByVal keyword.

ByVal Keyword

Used to pass a parameter by value.

Syntax

ByVal *parameter*

Part	Type	Description
parameter	Any Data Type	Parameter to be passed by Value.

Notes

ByVal can be used in the Method Declaration dialog box to denote that a parameter is to be passed by value. By Default, parameters are passed by value. Therefore, the **ByVal** keyword is optional.

Parameters passed by value are treated as local variables inside the method—just like variables that are created using the Dim statement. This means that you can modify the values of the parameters themselves rather than first assigning the parameter to a local

variable. For example, if you pass a value in the parameter "x", you can increment or decrement the value of x rather then assigning the value of x to a local variable that is created using Dim.

See Also ByRef keyword.

Call Statement

Enables you to call a function without handling the value returned by the function.

Syntax **Call *functionName***

Part	Description
functionName	Any function.

Notes Use the **Call** statement only when you want to call a function without using the value that it returns. This enables you to have the function perform its job without declaring an extra local variable to store the function's result.

Example The following calls the SelectColor function without using its (Boolean) result.

```
Dim c  as Color
c=CMY(.35,.9,.6) //choose the default color shown in color picker
Call SelectColor(c,"Select a Color")
Rectangle1.FillColor=c
```

See Also Function statement.

Canvas Control 

Canvas controls are very useful for implementing your own graphical controls because you can use the Graphics class drawing commands or the Object2D classes to draw inside the Canvas region. It can also be used to display existing graphics, like the ImageWell.

Super Class RectControl

Because this is a RectControl, see the RectControl class for other properties and events that are common to all RectControl objects.

Properties

Name	Type	Description
AcceptFocus	Boolean	If True, the control can have the focus. It can accept the GotFocus and LostFocus events and can display a focus ring if UseFocusRing is set to True. Default is False.
AcceptTabs	Boolean	If True and AcceptFocus is True, then pressing Tab triggers the KeyDown event for processing. If False, pressing the Tab key does not trigger the KeyDown event; pressing Tab triggers the LostFocus Event and selects the next object in the window that can accept the focus.
Backdrop	Picture	The Picture that will automatically be drawn into the area. Must be a PICT on Macintosh or BMP on Windows unless the user has QuickTime installed.
Graphics	Graphics	Allows access to all the Graphics methods for drawing into the area. Whatever objects are drawn using the Graphics methods are drawn after the Backdrop image is drawn, in a separate layer. This allows you to overlay objects on top of the Backdrop image.
UseFocusRing	Boolean	If True, the Canvas indicates that it has the focus with a ring around its border; if False, the appearance of the object does not change when it has the focus. Setting this property to True has no effect on Windows. Default is True.

Events

Name	Parameters	Description
EnableMenuItems		The Canvas control has received the focus. Menu handlers are invoked if the Canvas control has the focus.
GotFocus		The Canvas control has received the focus.
KeyDown	Key as String	The key that has been passed to the event was pressed while the Canvas control has the focus.
LostFocus		The Canvas control has lost the focus.
MouseDown(function)	x as Integer, y as Integer	The mouse button was pressed inside the canvas region at the location passed in to x,y. Return True if you are going to handle the MouseDown.
MouseDrag	x as Integer, y as Integer	The mouse button was pressed inside the Canvas control and moved (dragged) at the location local to the control passed in to x,y. This event will not occur unless you return True in the MouseDown event.
MouseUp	x as Integer, y as Integer	The mouse button was released inside the Canvas region at the location passed in to x,y. This event will not occur unless you return True in the MouseDown event.

Name	Parameters	Description
Paint	g as Graphics	The area needs to be redrawn, such as when it has been covered by another window and then uncovered. The parameter passed gives you access to the Graphics class methods for drawing. Use the Paint event, rather than the Open event, if you want objects drawn with the Graphics class methods to be persistent.

Methods

Name	Parameters	Description
Scroll	DeltaX as Integer, DeltaY as Integer, [Left as Integer], [Top as Integer], [Width as Integer], [Height as Integer], [ScrollControls as Boolean]	*DeltaX* and *DeltaY* are the number of pixels to scroll horizontally and vertically relative to the current position of the picture in the Canvas control. Positive values scroll right and down while negative values scroll left and up. The *Left*, *Top*, *Width,* and *Height* parameters specify the portion of the picture to be scrolled. If these are not passed, the entire picture will be scrolled. ScrollControls indicates whether controls positioned on top of the Canvas control should be scrolled as well. ScrollControls is True by default.

Notes

Coordinates passed to Canvas events are local to the Canvas object.

To use the Scroll method to scroll the picture in a Canvas control, you need to store the last scroll value for the axis you are scrolling so you can use this to calculate the amount to scroll. This can be done by adding a property to the window that contains the Canvas control or by creating a new class based on the Canvas control that contains properties to hold the last X scroll amount and last Y scroll amount.

If the ScrollControls parameter is True, any controls on top of the Canvas control will also be scrolled. This allows the implementation of a scrolling pane of controls.

Examples

This example draws a 3D rectangle with a raised look.

```
Sub Paint(g as Graphics)
  g.forecolor=RGB(255, 255, 255) //white
  g.drawline 1,1,canvas1.width,1
  g.drawline 1,canvas1.height-1,1,1
  g.forecolor=RGB(140,140,140) //dark gray
  g.drawline canvas1.width-1,2,canvas1.width-1,canvas1.height
  g.drawline 1,canvas1.height-1,canvas1.width,canvas1.height-1
  //fill in the light gray rectangle
  g.forecolor=RGB(239,239,239)
  g.fillrect 2,2,canvas1.width-3,canvas1.height-3
End Sub
```

This example assigns a picture that has been dragged into the Project Window to the Backdrop property:

```
Me.backdrop=OSLogo
```

You can then use the methods of the Graphics class to modify the picture in any way you like. For example, the following code in the object's Paint event handler adds a caption to the picture:

```
Me.Graphics.TextFont="Helvetica"
Me.Graphics.TextSize=14
Me.Graphics.ForeColor=RGB(255,255,255)
Me.Graphics.DrawString "Joe Bob",10,100
```

If you instead assigned the graphic to the BackDrop property using NewPicture, as shown below, you could manipulate the graphic at the pixel level using the RGBSurface property of the Picture object.

```
Me.backdrop=NewPicture(210,30,32)
Me.backdrop.Graphics.DrawPicture madeWithREALbasicLogo,0,0
```

See also the examples for the Control class, which gives an example of dragging from a Canvas control and the ImageWell class example, which show drag and drop to and from the ImageWell.Image property.

Simulating a Focus Ring on Windows

Unfortunately, a Focus Ring does not appear on Win32 builds when a Canvas control gets the focus, but the GotFocus and LostFocus events fire normally. You can easily use them to simulate a focus ring.

In the GotFocus event, use:

```
#If TargetWin32
 Me.Graphics.forecolor=HighlightColor //or FrameColor, whichever you wish
 Me.Graphics.DrawRect 0,0,me.width-1,me.height-1
#endif
```

In the LostFocus event, use:

```
#If TargetWin32
 Me.Graphics.forecolor=RGB(178,178,178) //gray
 Me.Graphics.DrawRect 0,0,me.width-1,me.height-1
#endif
```

The simulated focus ring looks like this:

See Also Graphics, Picture classes; NewPicture function.

CDbl Function

This function is the same as the Val function but is used when you need to pass a string that uses a character other than the period (.) as the decimal separator. It uses the character specified by the Numbers Control panel.

See the Val function for more information.

See Also CStr function, Str function, Val function.

Ceil Function

Returns the value specified rounded up to the nearest integer.

Syntax ***result*=Ceil (*value*)**

Part	Type	Description
result	Double	The ceiling of *value*.
value	Double	The value you want the ceiling of.

Notes The **Ceil** function returns the value passed to it rounded up to the nearest integer.

Examples This example uses the **Ceil** function to return ceiling of a number.

```
Dim d as Double
d=Ceil(1.234) //returns 2
```

See Also Floor function, Round function.

ChasingArrows Control 

Adds chasing arrows to a window.

Super Class

RectControl

Because this is a RectControl, see the RectControl for other properties and events that are common to all RectControl objects.

Notes

Use the Visible property of the control to show or hide it. When it is visible, it is animated, i.e., 'chasing.'

See Also

RectControl class.

Checkbox Control 

The standard checkbox button control.

Super Class

RectControl

Because this is a RectControl, see the RectControl for other properties and events that are common to all RectControl objects.

Properties

Name	Type	Description
Bold	Boolean	Applies the bold style to the Checkbox caption.
Caption	String	The Checkbox's label.
DataField	String	Relevant only if the CheckBox is used in conjunction with a DataControl to display the contents of a field in a database table. Name of a Field in the table referenced by DataControl.
DataSource	DataControl	The DataControl that references a table in a database whose fields are displayed. Use the DataField property to link a field to be displayed in a CheckBox. In the IDE, a popup menu of field names will appear in the Properties window. When setting in code, it takes a String. Assign it to the Name property of a DataControl. Note: Database fields can also be displayed using EditFields, ListBoxes, and StaticText controls.
Italic	Boolean	Applies the italic style to the caption.
TextFont	String	Name of the font used to display the button text.

Name	Type	Description
TextSize	Integer	Size of the font used to display the button text.
Underline	Boolean	Applies the underline style to the button caption.
Value	Boolean	Value of the checkbox when its owning window opens.

Events

Name	Parameters	Description
Action		The control has been clicked.
GotFocus		(Win32 only) The control has received the focus and has a selection marquee around its caption.
LostFocus		(Win32 only) The control has lost the focus.
MouseDown	x as Integer, y as Integer	The mouse button was pressed inside the control's region at the location passed in to x,y. Return True if you are going to handle the MouseDown. The Action event will not execute and the state of the object will not change.
MouseUp	x as Integer, y as Integer	The mouse button was released inside the control's region at the location passed in to x,y. This event will not occur unless you return True in the MouseDown event. The return value is ignored.

Note

If the Caption property contains an ampersand character, it will display only if it is preceded by another ampersand character. For example, if the caption should read "Bread & Butter", enter "Bread && Butter".

This is done to make Macintosh and Windows applications behave consistently. The ampersand is used to denote a keyboard accelerator on Windows See "Importing and Exporting Modules" on page 231 of the Developer's Guide.

Example

The following code in the CheckBox's Action event handler checks the value of the CheckBox.

```
If CheckBox1.value then
 EditField1.text="True"
else
 EditField1.text="False"
end if
```

See Also

RadioButton control.

Chr Function

Returns the character whose ASCII value is passed.

Syntax

***result*=Chr(*value*)**

Part	Type	Description
result	String	The character whose ASCII *value* was passed.
value	Integer, Single, or Double	The ASCII number of the character you want.

Notes

The **Chr** function returns the character whose ASCII value is specified. ASCII codes for Windows and Macintosh are shown in Appendix B of the pdf and printed versions of the Language Reference.

The **Chr** function will return a single byte string when running on single byte systems and return a double byte string when running on double byte systems. If you need to get a single byte string regardless of whether the system software is single or double byte, use the ChrB function.

Examples

These examples use the **Chr** function to return the characters whose ASCII values are specified.

```
Dim Tab,CR as String
Tab=Chr(9)  //returns a tab
CR=Chr(13) //returns carriage return
```

See Also

Asc, InStr, Left, Len, Mid, Right functions.

ChrB Function

Returns a single byte string whose ASCII value is passed.

Syntax

***result*=ChrB(*value*)**

Part	Type	Description
result	String	The single byte string whose ASCII *value* was passed.
value	Integer, Single, or Double	The ASCII number of the character you want.

Notes

The **ChrB** function returns a single byte string whose ASCII value is specified. **ChrB** should be used rather than Chr when *value* represents binary data.

If you need to get a single byte string when running on single byte system software and a double byte string when running on double byte system software, use the Chr function.

Examples

These examples use the **ChrB** function to return the characters whose ASCII values are specified.

```
Dim s as String
s=ChrB(32)  //returns a space
s=ChrB(13) //returns carriage return
```

See Also

AscB, Chr, InStrB, LeftB, LenB, MidB, RightB functions.

ClearFocus Method

Removes the focus from the control that has the focus, leaving no control with the focus.

Syntax

ClearFocus

Notes

Calling ClearFocus has the same effect as calling the Window class method, SetFocus. You can set the focus using the SetFocus property of the RectControl class. Please note that the SetFocus method of the RectControl class does nothing when the control it is being applied to is not capable of having the focus on the platform on which the code is running.

Example

The following code in the Action event of a PushButton removes the focus from the object in the window that has the focus.

```
ClearFocus
```

See Also

Window, RectControl classes.

Clipboard Class

Used to read and write information to and from the Clipboard.

Super Class

Object

Properties

Name	Type	Description
Picture	Picture	The picture on the Clipboard.
PictureAvailable	Boolean	Returns True if the Clipboard contains a picture.
Text	String	The text on the Clipboard.

Methods

Name	Parameters	Description
AddMacData	Data as String, MacType as String	Copies the binary data specified in the format specified to the Clipboard.
Close		Closes the Clipboard. The Clipboard must be closed within the event handler it was opened in or an error will occur.
MacData	MacType as String	Returns the binary data from the Clipboard as a string for the specified type.
MacDataAvailable	MacType as String	Returns True if the Clipboard contains data of the type specified.
SetText	String	Sets the Clipboard to the text passed.
TextAvailable		Returns True if the data on the Clipboard is text.

Notes

In order to read text from or write text to the Clipboard, you must create an object of type Clipboard and then access the Text or Picture properties of the Clipboard object you create. You can place a picture on the Clipboard by setting the Picture property. You can use the SetText and AddMacData methods to write text or binary data to the Clipboard.

Examples

This example gets the text on the Clipboard and copies it to a variable.

```
Dim c as Clipboard
Dim s as String
c=New Clipboard
s=c.text
c.close
```

This example puts text on the Clipboard.

```
Dim c as Clipboard
c=New Clipboard
c.text="The quick brown fox had a Macintosh G4!"
c.close
```

This example checks to see if the Clipboard is text data and if so, copies the contents of the clipboard to an EditField.

```
Dim c as Clipboard
c=New Clipboard
 If c.TextAvailable then
  EditField1.text=c.Text
 End if
c.close
```

This example copies a PICT image to the Clipboard:

```
Dim c as Clipboard
c=New Clipboard
If ImageWell1.image <> Nil then
 c.Picture=ImageWell1.image
 c.Close
else
 MsgBox "No picture is available!"
End if
```

CMY Function

Returns a Color based on the CMY (cyan, magenta, yellow) color model.

Syntax

***result*=CMY(*cyan*, *magenta*, *yellow*)**

Part	Type	Description
result	Color	An object that represents the color based on the *cyan*, *magenta*, and *yellow* values.
cyan	Double	The value of Cyan in the color (0-1).
magenta	Double	The value of Magenta in the color (0-1).
yellow	Double	The value of Yellow in the color (0-1).

Notes

The **CMY** function returns a Color based on the amounts of Cyan, Magenta, and Yellow passed. These amounts are represented by doubles between 0 and 1 and are stored

internally as three bytes. You can also use either the RGB or HSV models to assign a color.

Examples

This example uses the **CMY** function to assign a Color to the Fillcolor property of a Rectangle control.

```
rectangle1.fillcolor=CMY(.35,.9,.6)
```

See Also

Color data type; HSV, RGB, SelectColor functions.

Collection Class

Used to store a set of related items in a larger structure. The concept of a collection is similar to that of an array, except that each element in a collection can be a different data type. A collection can be thought of as an array of variants. Also, each element in a collection can be named. Elements in a collection can be referred to either by number or name.

The Dictionary class provides all the functionality of the **Collection** class and offers several advantages: With the **Collection** class, the time taken to locate an item is a function of the number of items in the **Collection** because the search is sequential. A Dictionary uses a hash table, making the time (relatively) independent of the number of items. It is designed for high-speed lookups. Also, the key parameter in a Dictionary is a Variant, but is a String in the **Collection** class. Therefore, we recommend that you use the Dictionary class rather than the **Collection** class whenever possible.

The **Collection** class is included mainly for backward compatibility and for conversion of Visual Basic projects.

Super Class

Object

Methods

Name	Parameters	Description
Add	Value as Variant Key as String	Value is added as the last element of the collection; if *Key* is provided, it may be used to refer to the element using the Item property.
Count		Number of elements in the collection. Returns an integer.
Item	Index as Integer OR key as String	Refers to an element of a collection. Returns a Variant. Index is 1-based.
Remove	Index as Integer OR key as String	Removes the specified element of a collection. Index is 1-based.

Example

The following example creates a collection, populates it with both string and numeric values, and displays each element in EditFields or a Canvas control (The picture "lois" has been added to the project). Note that a collection is much like a database record.

```
Dim c as Collection
c=New Collection
c.add 1,"ID"
c.add "Lois Lane", "Name"
c.add "Reporter", "JobTitle"
c.add 85000,"Salary"
c.add lois,"Picture"
EditField1.text=c.item("ID")
EditField2.text=c.item(2) //returns "Lois Lane"
EditField3.text=c.item("JobTitle")
EditField4.text=c.item("Salary")
Canvas1.backdrop=c.item("Picture")
```

If you want to use the Item or Remove methods to refer to an item, use parens around the parameter passed to the method. This is because the compiler doesn't know which data type you are passing. For example, use

```
c.remove("Name")
```

rather than

```
c.Remove "Name"
```

to remove the "Lois Lane" record from the collection.

See Also

Variant, Dictionary classes; VarType, Nil functions.

Color Data Type

A **Color** is an intrinsic data type that consists of three bytes that define the color. A color can be set using the RGB, HSV, or CMY models. Each set of properties is used with the appropriate function to specify a color. The values of any of the nine properties can then be read.

Properties

Name	Type	Description
Blue	Integer	The amount (0-255) of blue in the color (RGB)
Green	Integer	The amount (0-255) of green in the color (RGB).

Name	Type	Description
Red	Integer	The amount (0-255) of red in the color (RGB).
Hue	Double	The value of Hue (0-1) in the color (HSV).
Saturation	Double	The value of Saturation (0-1) in the color (HSV).
Value	Double	The value of Value (0-1) in the color (HSV).
Cyan	Double	The value of Cyan (0-1) in the color (CMY).
Magenta	Double	The value of Magenta (0-1) in the color (CMY).
Yellow	Double	The value of Yellow (0-1) in the color (CMY).

Notes

Colors are mostly used to assign colors to the properties of type **Color**. Use the RGB, HSV, or CMY functions to assign a color to an object or use the format:

```
&cRRGGBB
```

where *RR* is the RGB value of Red in hexadecimal, *GG* is the value Green in hexadecimal, and *BB* is the value of Blue in hexadecimal. To obtain the hex values easily, use the New Constant dialog box to create a color constant using its built-in Color Picker. Select the desired color visually. When you do so, REALbasic will calculate the hex values for the RGB model and insert them in the Value area of the dialog.

An easy way to create the desired color is to open the Colors palette and assign colors using the Color Picker. Just double-click on a blank item and choose a color with the Color Picker. Then drag a color from the Colors palette to the line of code to which you want to assign the color or the property in the Properties window. A drag includes both the (visual) color and its RGB values. You can also drag the color to the desktop or to a text editor that supports drag and drop.

You can use the VarType function to determine whether a property or a variable is a **Color**. If it is, VarType returns 16.

Examples

The following example uses the RGB model to set the forecolor property of a Canvas control and draw a square using the current forecolor. The code is placed in the Canvas control's Paint event.

```
Canvas1.graphics.foreColor=RGB(255,0,0)
Canvas1.graphics.drawrect 0,0,50,50
```

The following example assigns a color directly using the RGB color model. The RGB values must be specified in hexadecimal:

```
Canvas1.graphics.foreColor=&cFF0000
Canvas1.graphics.drawrect 0,0,50,50
```

The following example uses the CMY model to set the fillcolor of a Rectangle control, r.

```
r.fillColor=CMY(.1,.3,.5)
```

The following example inverts the fillcolor:

```
Dim c as Color
c=RGB(255-r.fillcolor.red,255-r.fillcolor.green, 255-r.fillColor.blue)
r.fillColor=c
```

See Also RGB, HSV, CMY, SelectColor, DarkTingeColor, FillColor, FrameColor, HighlightColor, LightBevelColor, LightTingeColor, TextColor, Variant, VarType functions.

Const Statement

Declares a value as a constant.

Syntax **Const *constantname = value***

Notes The **Const** statement can be used in place of the Dim statement followed by an assignment statement when you are sure that the value of the variable should not change within the method. Using **Const** instead of Dim provides a convenient way to manage such values.

A **Const** statement cannot be placed inside a conditional structure, such as an If statement, or a looping structure, but can be placed after lines of executable code.

Of course, constants declared in this manner are local to the method. You can also create global constants using a module, as described in the section "Adding Constants to Modules" on page 224 of the *User's Guide*. This feature provides a central location in which all the constants in the application are managed. Global constants can also be used to localize an application and to provide keyboard shortcuts (accelerators) on the Windows platform.

Examples The following constant is used to set a button's caption.

```
Const Accept="OK"
BevelButton1.caption=Accept
```

The following sets the value of "Pi" for the method.

```
Const Pi=3.14159265358979323846264338327950
```

See Also

Dim statement.

ContextualMenu Control

ContextualMenu controls are used to display and control contextual menus.

Super Class

Control

Because this is a Control, see the Control Class for other properties and events that are common to all Control objects.

Events

Name	Parameters	Description
Action	Item as String	The Item from the contextual menu was chosen.

Properties

Name	Type	Description
UseCCM	Boolean	If True, the Help item is displayed. If False, the Help item is omitted.

Methods

Name	Parameters	Description
AddRow	Item as String	Appends Item in a new row to the end of the list.
AddSeparator		Appends a separator line to the end of the list.
DeleteAllRows		Deletes all rows in the list.
Open		Displays the contextual menu.

Notes

Contextual menus are created in the same way PopupMenu controls are. If you have a static set of items for a contextual menu, you can build the menu in the ContextualMenu's Open event handler. If the items in the menu change, you may want to build the contextual menu on the fly—after testing IsCMMClick—in a MouseDown or MouseEnter event handler. To display the contextual menu, call its Open method.

There are two choices for implementing contextual menus when you have more than one control in a window that requires a contextual menu. The simplest method is to add one ContextualMenu control for each control that requires a contextual menu. The second method would require you to add a property of type Control or RectControl to the window and store a reference to the object that requires the display of a contextual menu. Then a single ContextualMenu control can be used to display and control the

contextual menus because it can use the property you have added to determine which object the user is clicking. This second method is a bit more complex but will be more efficient when you have several controls in a window that all require the same contextual menu.

Example

The following code in a ContextualMenu's Open event handler populates the control:

```
Dim s as String
Dim i,last as Integer
s="Spring,Summer,Fall,Winter"
last=CountFields(s,",")
For i=1 to last
 Me.addRow NthField(s,",",i)
Next
```

The following code in a Canvas control's MouseDown event handler displays the contextual menu when the user holds down the Control key and the mouse button:

```
If IsCMMClick then
 ContextualMenu1.open
 Return True
End if
```

When the user chooses a menu item, the ContextualMenu's Action event handler runs.

See Also

Control class; IsCMMClick function.

Control Class

Control classes include the Line, NotePlayer, RectControl, ContextualMenu, and Timer controls.

Super Class

Object

Properties

Name	Type	Description
Handle	Integer	Returns a handle to the control. This property provides identical functionality to MacControlHandle, but the name change reflects that it is now cross-platform. Any visible Win32 control should have a handle.

Name	Type	Description
MacControlHandle	Integer	Returns handle to control. Can be used by toolbox programmers to control the Control directly. This property appears in REALbasic 3.x for compatibility with earlier releases only. It will not be supported in future releases. You should recode to use Handle instead.
MouseX	Integer	The X coordinate of the mouse (pixels). Measured from the top-left corner of the window.
MouseY	Integer	The Y coordinate of the mouse (pixels). Measured from the top-left corner of the window.
Name	String	The name of the control.
TabPanelIndex	Integer	If the control has been placed on a panel of a TabPanel control, it returns the panel number. The first panel is numbered 1. If the control is not on a TabPanel, it returns zero. This property is readable, but you cannot use it to move a control to another panel. **Note:** This property may be replaced in future releases with another way of determining the location of controls. Please be advised that you may need to modify your code at a later date if you use TabPanelIndex.

Events

Name	Parameters	Description
Close		The window is about to close.
Open		The window is about to open.

Methods

Name	Parameters	Description
Close		Closes a control. Closing a control permanently removes the control from memory, making it impossible to access. You can close both non-indexed controls and indexed controls. When you close an indexed control, the indexes for the remaining controls will shift downward so that the indexes start with zero and are consecutive.

Notes

Changing the Cursor

The MouseCursor property controls which cursor will be displayed while the mouse is within the control, assuming that the MouseCursor properties of the Application and parent Window classes are Nil. You can, for example, assign different MouseCursors to different controls within a window and the shape of the pointer will change whenever it is over a particular control.

If you also need custom cursors at the Window or Application levels, you need to manage them along the lines described in the section on the MouseCursor class.

There are three built-in cursors: WatchCursor, IBeamCursor, and ArrowCursor. To add other cursors to your project, do this:

1. Launch ResEdit.
2. Create a new resource file.
3. Add a resource to the file of type CURS.
4. Use ResEdit's editor to create the look of the cursor you want or paste a cursor from another CURS resource. Create one cursor per CURS resource ID.
5. Save the new resource file.
6. Quit ResEdit.
7. Drag the new resource file into your Project window.

You can then access, for example, CURS resource ID 1027 by putting the following statement in the control's MouseEnter event handler:

```
Me.MouseCursor=App.ResourceFork.GetCursor(1027)
```

This will change the pointer to the specified CURS resource when the user moves the pointer over the control.

Custom Cursors in Windows

If you need to use custom cursors in the Windows build of your application, you can do so using a slightly modified version of this technique. This technique is also fully compatible with Macintosh builds.

1. With a resource editor, create a CURS resource that contains only one cursor. Give the resource file a name that indicates the type of cursor contained in the resource file.
2. Repeat this process for each custom cursor.
3. Drag all the resource files into the Project Window. These special resource files display a Pointer icon in the Project Window:

4. You can then access each custom cursor by name, e.g.,

```
Me.MouseCursor=Pen
```

Dragging

There are three possible items a user can drag from a control: text, a picture, and/or a FolderItem. In order for the user to be able to drag, the control must (during the appropriate event handler) create a DragItem object. The appropriate event handler depends on the type of object. For example, the appropriate event handler for a Canvas control is the MouseDown event handler since the user must be holding down the mouse button to drag.

The DragItem object represents the data that the user is dragging. DragItem objects have properties to hold text, pictures, and FolderItems. One or more of these properties must be populated with the values the user wishes to drag. When you create a new DragItem object using the NewDragItem method, you can specify the drag rectangle that should appear as the user drags from the control.

EditFields and ListBoxes have implicit dragging built-in to them. This means that they will create a new DragItem and drag rectangle for you. EditFields automatically populate the Text property of the DragItem with the text the user is dragging. ListBoxes have a DragRow event handler where you can populate the DragItem with the data to be dragged.

Dropping

Before a control will accept an item being dropped on it, you must call the AcceptFileDrop, AcceptPictureDrop, and/or AcceptTextDrop methods to indicate the drop types that will be allowed. This is usually done in the control's Open event handler. When an acceptable item is dropped on the control, the control's DropObject event handler will execute and will be passed the DragItem that has been dropped on the control. At that point you can use the DragItem's methods to determine what kind of data is available and access it. If you are only allowing a particular type of drag data (text for example) then there would be no need to test the type of data that is being dragged. You can simply get the text from the Text property of the DragItem. See the "DragItem Class" on page 125 for more information and examples.

Examples

This example shows how the DropObject event handler of an EditField (named EditField1 in this example) would be coded to handle one or more SimpleText documents or some text being dropped on it:

```
Sub DropObject(Obj as DragItem)
  If Obj.TextAvailable then
   EditField1.Text=Obj.Text
  ElseIf Obj.FolderItemAvailable Then
   Do
    //Load the styled text from the document
    //into EditField1
    Obj.FolderItem.OpenStyledEditField EditField1
   Loop Until Not Obj.NextItem
  End if
End Sub
```

See Also

Line, NotePlayer, RectControl, and Timer controls; Object class.

ControlPanelsFolder Function

Used to access the Control Panels folder. On Windows, a call to ControlPanelsFolder returns Nil.

Syntax

***result*=ControlPanelsFolder**

Part	Type	Description
result	FolderItem	A FolderItem that represents the Control Panels folder.

Notes

Use the **ControlPanelsFolder** function to access the Control Panels folder.

The **ControlPanelsFolder** function provides a way to access the Control Panels folder that will work under different language systems, and will continue to work even if Apple reorganizes the System folder. If your application will be used on Windows, you can check the value of the TargetWin32 global constant before calling **ControlPanelsFolder**. Since ControlPanelsFolder returns Nil under Windows, it doesn't hurt to test for a Nil value before trying to access the FolderItem returned by **ControlPanelsFolder**.

Examples

See the AppleMenuFolder function for an example.

See Also

DesktopFolder, ExtensionsFolder, FontsFolder, ApplicationSupportFolder, PreferencesFolder, ShutDownItemsFolder, StartupItemsFolder, SystemFolder, TemporaryFolder, TrashFolder functions.

ConvertEncoding Function

Provides a quick way to convert a string of known encoding to some other encoding, without having to create a TextConverter object.

Syntax

***result*=ConvertEncoding(*Str*, *NewEncoding*)**

Part	Type	Description
result	String	The converted string using the *NewEncoding* TextEncoding.
str	String	The string to be converted.
newEncoding	TextEncoding	The encoding to be used in the conversion.

See Also

TextEncoding, TextConverter classes.

Cos Function

Returns the cosine of the given angle.

Syntax

***result*=Cos (*value*)**

Part	Type	Description
result	Double	The cosine of *value*.
value	Double	The value (in radians) you want the cosine of.

Notes

The **Cos** function returns the cosine of the angle (in radians) passed to it. If the angle is in degrees, multiply it by PI/180 to convert it to radians.

Examples

This example uses the **Cos** function to return the cosine of an angle.

```
Dim d as Double
Const PI=3.14159265358979323846264338327950
d=Cos(45*PI/180) //returns 0.707
```

See Also

Acos function.

CountFields Function

Returns the number of values (fields) in the string passed that are separated by the separator character passed.

Syntax

***result*=CountFields(*source*, *separator*)**

Part	Type	Description
result	Integer	The number of values in *source* that are separated by *separator*.
source	String	The original string.
separator	String	The character that separates the values in *source*.

Notes

The **CountFields** function is useful for reading columns of data from a text file where the columns (fields) are delimited with a specific character.

If the separator is not found within *source*, **CountFields** returns 1. If source is null, **CountFields** returns zero.

Examples

The example below returns 5.

```
Dim count as Integer
Dim s as String
s="Dan*Smith*11/22/69*5125554323*Male"
count=CountFields(s, "*")
```

The following example returns three because it counts the null "field" after the (unnecessary) final field delimiter.

```
Dim count as Integer
Dim s as String
s="Dan*Smith*"
count=CountFields(s, "*")
```

See also the example that illustrates how to populate a PopupMenu control.

See Also

TextInputStream object example; NthField function.

CriticalSection Class

Used to protect a critical section of code in a multithreaded environment.

Super Class

Object

Methods

Name	Parameters	Description
Enter		Start of critical section of code
Leave		End of critical section of code

Notes

A **CriticalSection** is very similar to a Semaphore, except the critical section is usually used to protect a critical section of code. The Enter/Leave methods of a critical section work similarly to Signal/Release — the main difference being that **CriticalSection**.Enter can be called multiple times by the same thread (balanced by an appropriate number of Leave calls), while Semaphore.Signal can only be called once.

See Also

Semaphore class.

CStr Function

This function is the same as the Str function but is used when you need to pass a number that uses a character other than the period (.) as the decimal separator. It uses the character specified in the Numbers Control panel.

See the Str function for more information.

See Also

CDbl, Str, Val functions.

CurveShape Class

Used for drawing lines and curves in a vector graphics environment.

Super Class

Object2D

Properties

Name	Type	Description
ControlX	Double	Parameter is Index as Integer. A zero-based array of control points (horizontal position)
ControlY	Double	Parameter is Index as Integer. A zero-based array of control points (vertical position).
Order	Integer	The number of off-curve control points that are used. It is one of the following values: 0—A straight line from x,y to x2,y2 is used. 1—A quadratic Bezier curve is drawn using one control point. 2—A cubic Bezier curve is drawn using two control points.
Segments	Integer	Number of straight line segments to use to approximate a curve. The default value of zero tells REALbasic to choose a 'sensible' number of segments.
X2	Double	The horizontal position of the end of the line or curve.
Y2	Double	The vertical position of the end of the line or curve.

Notes

For CurveShapes, the default value for the Border property is 100 and the default value for Fill is 0.

When you set Order to zero, you get a straight line from x,y to x2, y2. You use control points when you want to 'bend' the line in a particular way. When you set Order to 1, the line bends towards ControlX(0),ControlY(0) on its way from X,Y to X2,Y2. This is what happens in the example. The further away the control point is, the more the line will deviate in that direction.

If Order is set to 2, there are two control points. The line will bend first towards ControlX(0),ControlY(0), and then towards ControlX(1),ControlY(1). At the end points, the curve points directly at the associated control point. The curve is guaranteed to always stay within the polygon formed by the endpoints and the control points (if any).

Example

The following method, when placed in the MouseDown event of a window, draws a simple little curve when the user presses the mouse button. The negative value of

ControlY(0) places the control point above the imaginary straight line from x,y to x2,y2.

```
Dim c As CurveShape
c=New CurveShape
c.controlx(0)=120
c.controly(0)=-40
c.order=1
c.x=10
c.y=10
c.x2=250
c.y2=10

Graphics.DrawObject c,x,y
```

The curve looks like this:

See Also FigureShape, FolderItem, Group2D, Graphics, Object2D, OvalShape, Picture, PixmapShape, RectShape, RoundRectShape, StringShape classes.

DarkBevelColor Function

The currently selected Appearance Manager color for drawing dark lines in dividing lines and group boxes.

Syntax ***result*=DarkBevelColor**

Part	Type	Description
result	Color	The color used for drawing the dark lines in dividing lines and group boxes.

Notes This value is useful when you are using Canvas controls to create custom controls that you want to be Appearance Manager savvy. When drawing controls like dividing Lines and GroupBoxes, use this color for the dark portions of the object (usually the right and bottom sides of the object).

This value can be changed by the user on the fly using the Kaleidoscope Control Panel, so you should access this value during the Paint event handler rather than storing the value.

See Also DarkTingeColor, FillColor, FrameColor, HighlightColor, LightBevelColor, LightTingeColor, TextColor, RGB, CMY, HSV, functions; Color data type.

DarkTingeColor Function

The currently selected Appearance Manager color for drawing dark lines inside frames on the bottom and right sides.

Syntax

***result*=DarkTingeColor**

Part	Type	Description
result	Color	The color used for drawing the dark lines inside frames on the bottom and right.

Notes

This value is useful when you are using Canvas controls to create custom controls that you want to be Appearance Manager savvy. When drawing beveled objects, use this color for the dark portions of the object (usually the right and bottom sides of the object). In a CheckBox control, this color is used to draw the dark lines that give it the beveled appearance just inside the checkbox on the right and bottom sides.

This value can be changed by the user on the fly using the Kaleidoscope Control Panel, so you should access this value during the Paint event handler rather than storing the value.

See Also

DarkBevelColor, FillColor, FrameColor, HighlightColor, LightBevelColor, LightTingeColor, TextColor functions.

DataAvailableProvider Interface Class

Used to program custom object bindings.

Properties

Name	Type	Description
dataAvailable	Boolean	True if data is available

See Also

DataNotifier Interface class.

Database Class

A Database object represents an open database that can be accessed by REALbasic's "front-end" database commands. You can create or open a REAL database using the REALDatabase class. Use the ADSP4DServer, TCPIP4DServer, ODBCDatabase, OpenOpenBaseDatabase, MySQLDatabase, or PostgreSQLDatabase classes to return Database objects.

Except for the REAL database, you need to install the appropriate database plug-in in the REALbasic Plugins folder to use the other data sources. Database plug-ins may be updated more frequently than the REALbasic application itself. You can obtain the most current versions of all database plug-ins at www.realsoftware.com.

FrontBase can also be accessed using third-party plug-ins. The FrontBase plug-in is free and is available on the REALbasic CD.

The Standard version of REALbasic supports all database functionality but limits you to databases of no more than 50 records and does not include the ability to build standalone database applications.

Super Class

Object

Properties

Name	Type	Description
DatabaseName	String	The database name to open.
Error	Boolean	True if an error is returned from the database engine.
ErrorCode	Integer	Error code returned from database engine. Error codes and error messages are different for each engine.
ErrorMessage	String	Text of error message returned from the database engine. Error codes and error messages are different for each engine.
Host	String	The database host name or server to connect to.
Password	String	The password required for access to the database.
UserName	String	The username required for access to the database

Methods

Name	Parameters	Description
Close		Closes the database.
Commit		Commits (saves) changes to records. If you quit the application after making changes to a RecordSet, REALbasic issues an implicit Commit. Use Commit and Rollback to manage transactions.

Name	Parameters	Description
Connect		Connects to the database server and opens the database for access. Returns a Boolean. Before proceeding with database operations, test to be sure that Connect returns True.
FieldSchema	TableName as String	Returns a RecordSet with a information about all fields in the table. See notes, below. In versions 4.0 and earlier, it returned a DatabaseCursor.
InsertRecord	TableName as String, Data as DatabaseRecord	Inserts *Data* as the last row of *TableName*.
Property	Name as String	Returns the database property specified by *Name* from the data source. Currently supported only for 4D Server and for getting the connection string. See example.
Rollback		Cancels a set of changes to records.
SQLSelect	SelectString as String	The SQL Select statement. Returns a RecordSet. In versions 4.0 and earlier, it returned a DatabaseCursor.
SQLExecute	ExecuteString as String	The SQL Execute statement.
TableSchema		Returns a RecordSet with a list of all tables in the database. See notes, below. In versions 4.0 and earlier, it returned a DatabaseCursor.

Notes

Connecting to a data source

The following new subclasses of the Database class were added in version 4.5 of REALbasic. The corresponding functions in earlier versions of REALbasic are obsolescent

Class	Description
ADSP4DServer	Use instead of Open4DDatabasebyADSP
MySQLDatabase	Provides support for MySQL databases; this functionality was previously available only via third-party plug-ins.
ODBCDatabase	Use instead of OpenODBCDatabase
OpenBaseDatabase	Use instead of OpenOpenBaseDatabase
PostgreSQLDatabase	Use instead of OpenPostgreSQLDatabase
REALDatabase	Use instead of OpenREALDatabase and NewREALDatabase
TCPIP4DServer	Use instead of Open4DDatabasebyTCPIP

The functions listed above are included in the current version of REALbasic for compatibility, but we recommend using the new subclasses of the Database class. There are two major advantages:

- Using this scheme, we can provide functionality that is unique to a certain database; some database classes support different things.

- It is now possible to subclass the Database class and do something useful with it. This was impossible in the past since all the functions used to open a data source return a Database object.

  When establishing a connection to any data source, do a test like this before proceeding:

```
Dim db as Database
.
If db.Connect() then
//connection is successful
else
//connection failed
end if
```

  In general, you will use the Username, Host, and Password properties of the Database class to establish the connection. Specific requirements are different for different data sources.

FieldSchema and TableSchema

FieldSchema returns a RecordSet with five fields: ColumnName (String), FieldType (Integer), IsPrimary (Boolean), NotNull (Boolean), and Length in bytes (Integer) for Text fields. *FieldType* uses the following values. Not all data sources support all these field types. Also, some field types have implementation-specific features, such as numeric precision and maximum length. Consult the documentation for your data source.

Table 1: Field Types returned by FieldSchema

FieldType	Value	Description
Null	0	Denotes the absence of any value, i.e., a missing value.
Byte	1	Stores the byte representation of a character string.
SmallInt	2	A numeric data type with no fractional part. The maximum number of digits is implementation-specific, but is usually less than or equal to INTEGER. The REAL database supports 2-byte smallints, which allow you to store values in the range of ±32,767. If you are using another data source, check the documentation of your data source.
Integer	3	A numeric data type with no fractional part. The maximum number of digits is implementation-specific. The REAL database supports 4-byte integers, which provide a range of ±2,000,000,000. If you are using another data source, check the documentation of your data source.
Char	4	Stores alphabetic data, in which you specify the maximum number of characters for the field, i.e., CHAR (20) for a 20 character field. If a record contains fewer than the maximum number of characters for the field, the remaining characters will be padded with blanks.

Table 1: Field Types returned by FieldSchema (Continued)

FieldType	Value	Description
Text or VarChar	5	Stores alphabetic data, in which the number of characters vary from record to record, but you don't want to pad the unused characters with blanks. For example, "VARCHAR (20)" specifies a VARCHAR field with a maximum length of 20 characters. The REAL database supports the VarChar field type, not the Text type. The REAL database uses a maximum length of 8,174 bytes for a VARCHAR field; if you specify a maximum length in a CREATE TABLE or ALTER TABLE statement, it is ignored.
Float	6	Stores floating-point numeric values with a precision that you specify, i.e., FLOAT (5). The REAL database uses 4-byte floats.
Double	7	Stores double-precision floating-point numbers. The REAL database uses 8-byte doubles.
Date	8	Stores year, month, and day values of a date. The year value is four digits; the month and day values are two digits.
Time	9	Stores hour, minute, and second values of a time. The hours and minutes are two digits. The seconds values is also two digits, may include a optional fractional part, e.g., 09:55:25.248. The default length of the fractional part is zero.
TimeStamp	10	Stores both date and time information. The lengths of the components of a TimeStamp are the same as for Time and Date, except that the default length of the fractional part of the time component is six digits rather than zero. If a TimeStamp values has no fractional component, then its length is 19 digits If it has a fractional component, its length is 20 digits, plus the length of the fractional component.
Currency	11	Stores a decimal amount with a Dollar sign.
Boolean	12	Stores the values of TRUE or FALSE.
Decimal	13	Stores a numeric value that can have both an integral and fractional part. You specify the total number of digits and the number of digits to the right of the decimal place, i.e., DECIMAL (5.2) specifies a decimal field that can contain values up to 999.99. DECIMAL (5) specifies a field that can contain values up to 99,999.
Binary	14	Stores code, images, and hexadecimal data. Consult the documentation of your data source for information on the maximum size of a Binary field.
Long Text (Blob)	15	Stores a text object. Consult the documentation of your data source for information on the maximum size of a Blob.
Long VarBinary (Blob)	16	Stores a binary object. The REAL database supports blobs of up to 2GB. If you are using another data source, check the documentation of your data source.
MacPICT	17	Stores a Macintosh PICT image.
Unknown	255	Unrecognized data type.

The values of IsPrimary and NotNull are set using the SQL Select statement that created the table.

TableSchema returns a RecordSet with one field, TableName (String).

The Property Method

The purpose of the Property method is to retrieve miscellaneous information from the data source. For example, if you are using 4th Dimension as the data source, you can pass "connectionString" to the Property method and it will return the connection string for the 4D Server database. It contains the name of the database. ConnectionString is supported by 4th Dimension but may not be supported by other data sources.

SQLSelect and SQLExecute Methods

You use the SQLSelect and SQLExecute methods to communicate with your data source via SQL commands. REALbasic currently supports a subset of SQL. The following table lists the supported SQL statements.

Statement	Example	Description
SELECT WHERE ORDER BY GROUP BY	SELECT * from Customers	Returns a group of rows, known as a *cursor* in the language of SQL. REALbasic uses the term RecordSet. You can specify the columns (fields), the table(s), search conditions, grouping, and sorting columns.
CREATE TABLE	CREATE TABLE invoices (id integer not null, Cust_ID integer not null, Amount double, Date date, primary key (id))	Creates a table and specifies the fields and their attributes.
ALTER TABLE	ALTER TABLE <tablename> ADD <field name> <field type> [NOT NULL]	Adds a field to an existing table and, optionally specifies it as Not Null.
UPDATE	UPDATE Customers SET City='Toldeo', Telephone='312 555-1212' WHERE Cust_ID='0121'	Updates existing records.
INSERT	INSERT INTO pubs (pub_id, pub_name) VALUES (481,"Exploring REALbasic")	Inserts a record into the specified table. All values for all columns must be specified. You cannot insert a record but define only some of the columns.
DELETE	DELETE FROM authors WHERE au_lname ="Marx"	Deletes the records that match the WHERE clause.
COMMIT	COMMIT	Commits changes (inserts, deletes, updates) to the database.
ROLLBACK	ROLLBACK	Cancels changes (inserts, deletes, updates) to the database.

Statement	Example	Description
Set functions: Min, Max, Avg, Count, Sum	Select Count (*) from Customers Select Sum(Total) from Invoice Select Name from Invoice where total=(select max(Total) from Invoice)	Returns calculated values from a group of rows defined by WHERE clause. Used in SELECT statement with optional WHERE clause.

SELECT Statement

The following is the syntax for a SELECT statement in REALbasic:

SELECT *columnlist* FROM *tablelist* WHERE {LIKE} *searchcondition* GROUP BY *groupingcondition* ORDER BY *sortcondition*

Argument	Description
columnlist	List of fields separated by commas. Using the asterisk (*) in place of the fieldnames retrieves all fields in table. If a column name has spaces in it, it must be surrounded by brackets, e.g., [first name] should be used for the field 'first name'. You also need to use brackets in this manner if the column name uses ACSII characters above 127. If *tablelist* refers to multiple tables, use the syntax *tablename.fieldname* to refer to a field.
tablelist	List of tables separated by commas. If a tablename has spaces in it or uses characters above ASCII 127, it must be surrounded by brackets, e.g., [Product Groups]
searchcondition	Expression that specifies a subset of the rows in the table or tables. Must evaluate to a Boolean whose value is True for the desired rows. If *tablelist* refers to multiple tables, use the syntax *tablename.fieldname* in *searchcondition*, i.e., Customers.Customer_ID=Invoices.Customer_ID. *Searchcondition* may use the following comparison symbols: **Comparison** **Symbol** Equality = Not equal <> Less than < Less than or equal to <= Greater Than > Greater Than or equal to >= You can create a compound *searchcondition* using the OR, AND, and NOT conjunctions, e.g. "Customers.Customer_ID=56 AND Invoices.Customer_ID=56". To use a SQL wildcard, use the LIKE operator, e.g., select * from Customers where name LIKE "Smith%". See "Wildcard Searches" on page 89.
groupingcondition	Field or fields on which you wish to group the rows in the cursor. If a GROUP BY clause is used, the rows are organized in groups defined by the fields in this clause. The groups appear in alphabetical order.

Argument	Description
sortcondition	Field or fields on which you wish to sort the individual rows in the RecordSet. Use ORDER BY rather than GROUP BY if the field is unlikely to define groups, such as Last Name. By default, the rows appear in ascending order. If you wish to use descending order, include the modifier DESC, i.e., 'ORDER BY invoices.date DESC'.

Wildcard Searches

The REAL database supports the two SQL wildcard characters, per cent, "%", and the underscore character, "_". The per cent wildcard stands for any number of characters and the underscore stands for any single character. Placing the % at the beginning or the end of the search string does 'ends with' and 'begins with' searches, respectively. Using the % on both sides of the search string specifies a 'contains' search. For example,

SELECT statement	Search Type
SELECT * from authors where au_lname LIKE "Jon%"	Begins with
SELECT * from authors where au_lname LIKE "%son"	Ends with
SELECT * from authors where au_lname LIKE "%man%"	Contains

The search criterion is case-sensitive.

The underscore character is the wildcard character for a any single character. For example,

SELECT * from authors where au_lname LIKE "B_r"

Returns "Bar", "Bor", etc.

The SQL LIKE operator is not supported for 4D. This is due to limitations in 4th Dimension. However, the user can get most of the functionality of LIKE using the equals operator and the "@", which is the wildcard character in 4D.

SELECT statement	Search Type
SELECT * from authors where au_lname = "smi@"	Begins with
SELECT * from authors where au_lname = "@smi"	Ends with
SELECT * from authors where au_lname = "@smi@"	Contains

Joins

You do relational operations ('joins') by specifying the tables to be joined in the SELECT statement's *tablelist* and indicating in the WHERE clause how the tables are to be joined, i.e., rows in the 'many' table whose foreign key matches the primary key in the 'one' table.The section "Joins" on page 94 contains several examples of simple one-to-many joins. Other, more esoteric types of joins are also possible. Please refer to a SQL reference book for detailed information on relational operations.

CREATE TABLE Statement

The CREATE TABLE statement creates a table structure on the current data source. It specifies the table name, field names, and field attributes. The syntax is:

CREATE TABLE *tablename* (*fieldname1 fieldtype {not null}*,....*FieldnameN fieldtype {not null}*, Primary key (*fieldname)*)

Argument	Description
TableName	Name of the new table.
FieldName	Name of a field.
FieldType	Data type. The REAL database supports the following data types: Smallint, integer, float, double, boolean, date, time, and long binary (blob). If you are using another vendor's data source, see their documentation on supported data types. ''Field Types returned by FieldSchema'' on page 85 lists supported field types.
Primary Key	The field whose values uniquely identify the row, i.e., the identifying field in the table for relational operations.
Not Null	Optional. If Not Null is used, the database will require a value for that field in every row, i.e., the value may not be missing.

Use the SQLExecute method of the Database class when using the CREATE TABLE statement.

ALTER TABLE Statement

Use the ALTER TABLE statement to add a column to an existing table.

The syntax is:

ALTER TABLE *tablename* ADD *fieldname fieldtype* [NOT NULL]

Argument	Description
tablename	Name of the existing table.
fieldtype	Name of the field to add.
FieldType	Data type. The REAL database supports the following data types: Smallint, integer, varchar, float, double, boolean, date, time, and long binary (blob). If you are using another vendor's data source, see their documentation on supported data types. ''Field Types returned by FieldSchema'' on page 85 provides information on supported data types.
Not Null	Optional. If Not Null is used, the database will require a value for that field in ever row, i.e., the field may not be missing.

UPDATE statement

The UPDATE statement changes existing records in a table. Its syntax is:

UPDATE *tablename* SET *fieldname1=expression1*,... *fieldnameN=expressionn* WHERE *searchcondition*

Argument	Description
TableName	Name of table containing fields to be updated.
FieldName	Field in *TableName*.
Expression	Value to be assigned to Field.
Searchcondition	Boolean expression that identifies the row or rows to be updated.

You can also perform updates without using the UPDATE statement using the Edit and Update methods.

INSERT Statement

The INSERT statement allows you to insert a complete record into a table. You specify the table's name, all the fields, and the value for each field. Its syntax is:

INSERT INTO *tablename* (*fieldname1*,…,*fieldnameN*) VALUES (*value1*,…,*valueN*)

Argument	Description
tablename	Name of the existing table.
fieldname1 to N	Names of the fields in *tablename*.
value1 to N	Values for each field in the row being added.

For example:

INSERT INTO authors (au_Id, au_fname, au_lname) VALUES ("4578","Johnny","Grey")

DELETE Statement

The DELETE statement removes one or more records specified by a WHERE clause. Its syntax is:

DELETE FROM *tablename* WHERE *searchcondition*

Argument	Description
tablename	Name of the table from which you are removing records.
searchcondition	Expression that specifies a subset of the rows in the table. Must evaluate to a Boolean whose value is True for the desired rows. *Searchcondition* may use the following comparison symbols: **Comparison** **Symbol** Equality = Not equal <> Less than < Less than or equal to <= Greater Than >> Greater Than or equal to >= You can create a compound *searchcondition* using the OR, AND, and NOT conjunctions, e.g. au_lname="Smith" AND au_fname="Alfred". To use a SQL wildcard, use the LIKE operator, e.g., select * from Customers where name LIKE "Smith%" See "Wildcard Searches" on page 89.

For example,

DELETE FROM authors WHERE au_lname="Smith" and au_fname="Jack"

deletes the Jack Smith record from the authors table.

COMMIT and ROLLBACK statements

The COMMIT statement is issued to conclude a transaction (one or more inserts, updates, and/or deletes) and saves the results of the transaction to the data source. The ROLLBACK statement cancels the transaction and, in effect, rolls the database back to its state prior to the start of the transaction.

A transaction is started when you connect to the data source and continues until you issue a COMMIT or ROLLBACK command. At that point, a new transaction is started.

The syntax of each statement simply consists of a call without any parameters, i.e.,

COMMIT

and

ROLLBACK

Set Functions

The Set functions apply to sets of rows in a table and return the results of an arithmetic calculation. You determine the number of rows in the set using a standard WHERE clause and include the Set function in a SELECT statement. If the WHERE clause is omitted, the function is computed on all the rows in the table.

Function	Description and Example
Count	Number of rows in the RecordSet that have nonmissing data on the fields specified in select statement. SELECT COUNT (Name, Phone) from Customers WHERE Zip="48070"
Min	Minimum value of the field specified in SELECT statement. Select MIN (Price) from Products
Max	Maximum value of the field specified in the SELECT statement. Select MAX (Population) from Cities
Avg	Computes the average value of the field specified in the SELECT statement. SELECT AVG (Price) from Products WHERE Product="Database"
Sum	Computes the total of the values in the field specified in the Select statement. SELECT SUM (Population) from States WHERE Region="NE"

Examples

Creating a REAL database

The following code creates an internal database and uses SQLExecute to create a table.

```
Dim db as REALDatabase
Dim f as FolderItem
Dim result as Boolean
f=New FolderItem("mydb")
db=New REALDatabase
db.databaseFile=f
result=db.CreateDatabaseFile
If db.Connect() then
 db.SQLExecute("create table invoices(id integer not null, Cust_ID integer not
null, Amount double, Date date, primary key (id))")
else
 MsgBox "Database not created"
end if
```

The following example inserts a row in this table:

```
dim r as DatabaseRecord
dim mybool as Boolean
dim mydate as Date
.
.
r=New DatabaseRecord
r.IntegerColumn("id")=4
r.IntegerColumn("Cust_ID")=2
r.DoubleColumn("Amount") =9.98
mybool=ParseDate("10/22/98",mydate)
r.DateColumn("Date") = mydate
db.InsertRecord("invoices",r)
```

RecordSets

The following example creates a RecordSet that contains all rows and columns for a particular customer sorted by amount:

```
dim rs as RecordSet
.
.
rs = db.SQLSelect("select * from invoices where cust_id=02 ORDER BY
amount DESC")
```

The following Select statement retrieves all rows and columns from the Customers table in which the customer's last name begins with the letter 'L.'

```
rs=db.SQLSelect("select * from Customers WHERE customer.name LIKE 'L%'")
```

This Select statement retrieves all rows and columns in which the customer's last name does not begin with the letter 'L.'

```
rs=db.SQLSelect("select * from customer WHERE customer.name NOT LIKE
'L%'")
```

The following example returns a RecordSet that contains information on the fields in the customers table:

```
Dim rs as RecordSet
dim dbFile as FolderItem
dim db as Database
.
.
dbFile = GetFolderItem("test.rdb")
db=OpenREALDatabase(dbFile)
rs = db.FieldSchema("customers")
```

Joins

A join retrieves data from two or more tables. A join almost always uses a WHERE clause that specifies the rows to be included in the RecordSet. The WHERE clause usually refers to primary and/or foreign key fields in the tables being joined. The most common type of join retrieves one or more rows in one table that are logically linked to each row in a different table.

Consider the following simple tables:

ID	Name	Phone
1	Lois Lane	416 555-1212
2	Jimmy Olson	416 555-7510
3	Perry White	416 555-8190
4	Lana Lang	416 555-7000

ID	Cust_ID	Amount	Date
1	1	46.95	1/10/99
2	1	173.98	11/10/98
3	1	3.78	10/17/98
4	2	9.98	10/22/98
5	2	109.98	7/4/98

The second column in the Invoices table identifies an individual in the Customers table. The customers.ID field is the Customers table's primary key and the Invoices.Cust_ID field is the foreign key for the join. The following code:

```
Dim rs as RecordSet
.
rs = db.SQLSelect("select customers.name,customers.phone, invoices.amount,
invoices.date from invoices,customers WHERE
customers.id=invoices.Cust_ID")
```

produces the following database RecordSet:

Name	Phone	Amount	Date
Lois Lane	416 555-1212	46.95	1/10/99
Lois Lane	416 555-1212	173.98	11/10/98
Lois Lane	416 555-1212	3.78	10/17/98
Jimmy Olson	416 555-7510	9.98	10/22/98
Jimmy Olson	416 555-7510	109.98	7/4/98

The following Select statement retrieves the purchase amounts for the customer whose ID is 1, i.e., the first three rows of the RecordSet shown above:

```
rs = db.SQLSelect("select customers.name,customers.phone, invoices.amount,
invoices.date from invoices,customers WHERE customers.id =invoices.Cust_ID
and invoices.Cust_ID=1")
```

Set Functions

The following Select statement sums the Amount column for the customer with ID=1:

```
rs=db.SQLSelect("select customers.name,SUM (invoices.amount) from
invoices,customers where customers.id =invoices.Cust_ID" and customers.id =1")
```

This statement produces the following RecordSet:

Name	Amount
Lois Lane	224.21

Retrieving the Connection String from 4D Server

You can use the Property method to obtain the connection string from 4D Server. Once you have the connection string, you can access the database directly with

Open4DDatabasebyADSP or Open4DDatabasebyTCPIP rather than selecting it from the dialog box presented by Select4DDatabaseByADSP.

```
Dim db as Database
Dim theString as String
db=Select4DDatabaseByADSP ("getting the connectionstring","Dave","Gorilla")
If db.Connect() then
 theString=db.Property("connectionstring")
end if
db.close
```

See Also

DatabaseRecord DatabaseCursor, RecordSet, DatabaseField, ADSP4DServer, MySQLDatabase, ODBCDatabase, OpenBaseDatabase, PostGRESQLDatabase, REALDatabase, TCPIP4DServer classes; NewREALDatabase, OpenREALDatabase, Open4DDatabasebyADSP, Open4DDatabasebyTCPIP, OpenCSVCursor, OpenDBFCursor, OpenDTFDatabase, OpenODBCDatabase, OpenOracleDatabase, OpenPostgreSQLDatabase, OpenOpenBaseDatabase, Select4DDatabaseByADSP, SelectODBCDatabase functions.

DatabaseCursor Class

Used to create and manage database cursors. This class has been superseded by RecordSet and is maintained for compatibility with earlier versions of REALbasic. Please use the RecordSet class instead. Functions that returned a **DatabaseCursor** in earlier versions of REALbasic now return a RecordSet.

Super Class

Object

Properties

Name	Type	Description
BOF	Boolean	Beginning of the cursor
EOF	Boolean	End of the cursor
FieldCount	Integer	Number of fields in the cursor

Methods

Name	Parameters	Description
Close		Closes an open cursor. If you quit the application without calling Close, REALbasic does an implicit Close.
MoveFirst		Moves the record pointer to the first record in the cursor. Supported only for the REALdatabase data source.

Name	Parameters	Description
MoveLast		Moves the record pointer to the last record in the cursor. Supported only for the REALdatabase data source.
MoveNext		Moves the record pointer to the next record in the cursor. Supported for all data sources.
MovePrevious		Moves the record pointer to the previous record in the cursor. Supported only for the REALdatabase data source.
Field	Name as String	Returns value of the field in the row the record pointer is pointing to.
IdxField	Index as Integer	1-based array. Use to refer to i^{th} field in the cursor.
Edit		Call prior to performing modifications to the current record.
Update		Call to update the cursor to reflect changes to the record the record pointer is pointing to.
DeleteRecord		Deletes the record in the cursor that the record pointer is pointing to.

Notes

In the jargon of SQL, a *cursor* is a group of records. In this context, the word does not refer to the mouse pointer or the insertion point in a word processing document. This is how the word cursor has found its way into REALbasic. For example, the meaning of the term 'cursor' in the context of the OpenCSVCursor and OpenDFBCursor functions is completely different from its meaning in the context of the MouseCursor class.

REALbasic uses the term *recordset* to avoid this confusion. But, when you start reading SQL reference books, you will encounter the word 'cursor.' Keep in mind that they are just talking about a group of records in a SQL database—not your mouse pointer.

Although **DatabaseCursor** remains in the REALbasic language, you should recode to use RecordSet since functions that previously returned a **DatabaseCursor** now return a RecordSet.

See Also

Database, DatabaseRecord, RecordSet, DatabaseField classes; NewREALDatabase, OpenREALDatabase, Open4DDatabasebyADSP, Open4DDatabasebyTCPIP, OpenCSVCursor, OpenDBFCursor, OpenDTFDatabase, OpenODBCDatabase, OpenOracleDatabase, OpenPostgreSQLDatabase, OpenOpenBaseDatabase, Select4DDatabaseByADSP, SelectODBCDatabase functions.

DatabaseCursorField Class

Was used to obtain the values of database fields in a record in a database table. This class has been superseded by the DatabaseField class.

Super Class

Object

Properties

Name	Type	Description
DateValue	Date	Value of Date field for a row in the cursor.
DoubleValue	Double	Value of Double field for a row in the cursor.
IntegerValue	Integer	Value of Integer field for a row in the cursor.
JPEGValue	Picture	JPEG image for a field in a row in the cursor.
MacPictValue	Picture	Macintosh PICT image for a field in a row in the cursor.
Name	String	Name of field.
NativeValue	String	Binary form of value of field for a row in the cursor.
StringValue	String	Value of field for a row in the cursor as a string.

Methods

Name	Parameters	Description
GetString	Value as String	Gets the StringValue property. When used to get the string value of a date column, GetString returns the date in SQL date format which is *YYYY-MM-DD*, and *HH:MM:SS* for SQL time.
SetString	NewValue as String	Sets the StringValue property.

Notes

Assignments to the DateValue property store the time as well as the date, as with DatabaseRecords. When getting a value, the resulting Date object may now contain a time as well as a date. For fields of type Date, the time will be 00:00:00 (midnight); and for fields of type Time, the date will be January 1, 0001. Fields of type TimeStamp contain valid data for both the date and time.

See Also

Database, DatabaseCursor, DatabaseRecord, DatabaseField, RecordSet classes.

DatabaseField Class

Used to access the values of fields in a record in a database table. Use this class instead of the DatabaseCursorField class.

Super Class

Object

Properties

Name	Type	Description
BooleanValue	Boolean	Used to get and set the values of Boolean field types. Please use the Value property instead.
DateValue	Date	Used to get and set the values of Date field types. Please use the Value property instead.
DoubleValue	Double	Used to get and set the values of Double field types. Please use the Value property instead.
IntegerValue	Integer	Used to get and set the values of Integer field types. Please use the Value property instead.
StringValue	String	Used to get and set the values of String/Character field types. Please use the Value property instead.
Value	Variant	Used to get and set the value of a field of any data type. Set Value to Nil to set the field to NULL. The NULL value is currently supported in the MySQL, OpenBase, and PostgreSQL plug-ins.

Examples

The following example uses the Value property to set the values of fields of different data types.

```
Dim MyDB as Database
Dim rs as RecordSet
rs = MyDB.SQLSelect("select * from mytable")
rs.Edit
rs.field("MyName").value = Nil // NULL this field,
//does not work for all database plugins
rs.field("MyID").value = 23
rs.field("MyText").value = "Test"
rs.update
MyDB.Commit
```

The following method populates a ListBox with a RecordSet. It uses the Name and StringValue properties to obtain the fieldnames and values:

```
Sub populateCursor(d as RecordSet)
 dim i as Integer
 dim v as String

 ListBox1.deleteAllRows
 ListBox1.columncount = d.fieldcount
 ListBox1.addrow d.IdxField(1).Name
 ListBox1.cellBold(ListBox1.lastIndex, 0) = true
 for i = 2 to d.fieldcount
  ListBox1.cell(ListBox1.lastIndex,i-1) =d.IdxField(i).name
  ListBox.cellBold(ListBox.lastIndex, i-1) = true
 next
 While not d.eof
  v = d.IdxField(1).StringValue
   ListBox1.addrow v
   For i = 2 to d.fieldcount
    ListBox.cell(ListBox.lastIndex,i-1)=d.IdxField(i).StringValue
   next
  d.MoveNext
 wend
End Sub
```

See Also Database, RecordSet classes.

DatabaseQuery Control

The control by which you query databases. You ordinarily use a **DatabaseQuery** tool by binding it to the ListBox that is used to display the results of SQL queries. Alternatively, you can build databases without the **DatabaseQuery** control using only the commands in the Database theme.

Because its super class is Object, you can instantiate it with code, eliminating the need to actually add the control to a window.

Super Class Object

Properties

Name	Type	Description
Database	Database	The database that will be queried.
SQLQuery	String	The text of the SQL query to be run against *Database*.

Events

Name	Parameters	Description
QueryComplete		Runs when the query is finished.

Methods

Name	Parameters	Description
RunQuery		Executes the query defined by the *SQLQuery* property.

Examples

See the examples in the chapter "Creating Databases with REALbasic" in the *User's Guide*.

See Also

Database class; OpenREALDatabase, Open4DDatabasebyADSP, Open4DDatabasebyTCPIP, OpenCSVCursor, OpenDBFCursor, OpenDTFDatabase, OpenOpenBaseDatabase, OpenODBCDatabase, OpenOracleDatabase, Select4DDatabaseByADSP, SelectODBCDatabase functions.

DatabaseRecord Class

Used to create new Database records. The methods are used to populate the fields in a record. Call it once per column in the record.

Super Class

Object

Methods

Name	Parameters	Description
BlobColumn	Name as String	Parameter is column name. Sets the specified field to the specified blob.
BooleanColumn	Name as String	Parameter is column name. Sets the specified field to the specified boolean value.
Column	Name as String	Parameter is column name. Sets the specified field to the specified string value.

Name	Parameters	Description
DateColumn	Name as String	Parameter is column name. Sets the specified field to the specified date value.
DoubleColumn	Name as String	Parameter is column name. Sets the specified field to the specified double value.
IntegerColumn	Name as String	Parameter is column name. Sets the specified field to the specified integer value.
JPEGColumn	Name as String	Parameter is column name. Sets the specified field to the specified JPEG image.
MacPictColumn	Name as String	Parameter is column name. Sets the specified field to the specified Macintosh PICT image.

Notes

Assignments to the DateColumn property now store the time as well as the date, to support the SQL TimeStamp and Time field types (as well as Date). Note that if the date part is January 1, 0001, then this is converts to SQL as only a time (e.g., "18:54:00"), whereas if the date is anything else, it converts to SQL in full form (e.g., "2000-5-30 18:54:00").

Example

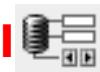
The following example creates a new REAL database, an employees table, and adds a record to the table.

```
Dim mybool as Boolean
Dim mydate as Date
Dim rec as DatabaseRecord
dbFile = GetFolderItem("test.rdb")
db = NewREALDatabase(dbFile)
db.SQLExecute("create table employees(id integer not null, name varchar,
jobtitle varchar, DOB date, Salary double, primary key (id))")
rec = New DatabaseRecord

rec.IntegerColumn("id") =1
rec.Column("name") = "Lois Lane"
rec.Column("jobtitle")="Pundit"
mybool=ParseDate("1Jan1950",mydate)
rec.DateColumn("DOB")=mydate
rec.DoubleColumn("Salary")=105000
db.InsertRecord("employees",rec)
```

See Also

Database, RecordSet, DatabaseCursor, DatabaseField, ADSP4DServer, TCPIP4DServer, REALDatabase, ODBCDatabase, OpenBaseDatabase, PostgreSQLDatabase classes; NewREALDatabase, OpenREALDatabase, Open4DDatabasebyADSP, Open4DDatabasebyTCPIP, OpenCSVCursor, OpenDBFCursor, OpenDTFDatabase, OpenODBCDatabase, OpenOracleDatabase, OpenOpenBaseDatabase, Select4DDatabaseByADSP, SelectODBCDatabase functions.

DataControl Control 

Allows the user to browse among records in a database table by clicking buttons. It consists of First Record, Previous Record, Next Record, and Last Record navigation buttons and a label that displays the Caption property. Built-in events fire when the user clicks any button. Methods allow you to navigate among rows programmatically.

A **DataControl** is designed to display data using EditFields, ListBoxes, StaticTexts, and PopupMenus. You can also create custom classes that are based on any of these classes and use them in conjunction with a **DataControl**.

Each of these controls (or their subclasses) has two properties that are specific to the **DataControl**: *DataSource* and *DataField*. DataSource accepts a DataControl object and DataField accepts the name of a field in the table to which the **DataControl** is linked. You set up a database interface by linking a **DataControl** to a database table and then linking each field whose values you wish to display to an EditField, ListBox, StaticText, or PopupMenu using the DataField and DataSource properties.

SuperClass

RectControl

Properties

Property	Type	Description
Caption	String	The text shown between the two sets of navigation buttons.
Cursor	RecordSet	The database cursor managed by the DataControl. Please use the RecordSet property instead.
Database	Database	The database to which the control is linked.
ReadOnly	Boolean	Set to True to prevent the user from adding, modifying, or deleting records. When True, the record navigation controls are disabled.
RecordLocked	Boolean	A value of True means that the record is locked, preventing changes to the record; False indicates that it is unlocked.
RecordSet	RecordSet	The RecordSet (cursor) managed by the DataControl.
SQLQuery	String	SQL query used to obtain a database cursor.
TableName	String	Name of the table in *Database* whose records will be displayed by the controls linked to the DataControl.

Events

Name	Parameters	Description
Delete		Occurs after the record is validated for deletion. The record is already deleted at this point.
Insert		Occurs after the new record is validated but before data aware fields are cleared.
MoveFirst		Occurs when the First Record button has been pressed but before moving to the first record. Returning T rue prevents the record pointer from moving to the first record.
MoveLast		Occurs when the Last Record button has been pressed but before moving to the last record. Returning True prevents the record pointer from moving to the last record.
MoveNext		Occurs when the Next Button has been pressed but before moving to the next record. Returning True prevents the record pointer from moving to the next record.
MovePrevious		Occurs when the Previous Button has been pressed but before moving to the previous record. Returning True prevents the record pointer from moving to the previous record.
Reposition	Index as Integer	Occurs after initialization and when the record pointer has moved to another record. The Index parameter indicates the row number after the reposition.
Update		Occurs after the record is validated for updating. The record is already updated at this point.
Validate	Action as Integer	Occurs when you have modified the current record via data aware EditFields. Returning False (the default) updates the record with your changes. Returning True prevents the update from happening. Action specifies what kind of action is being performed: 0=AddNew 1=Update 2=Delete

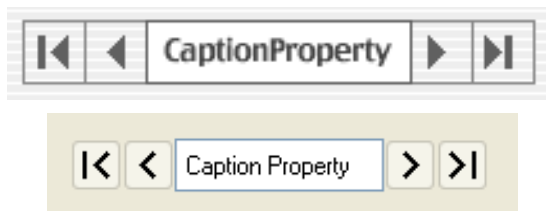
Methods

Method	Parameters	Description
Cursor		Returns a DatabaseCursor. Please use the RecordSet method instead.
Delete		Deletes the current record from the selection, leaving the record pointer at the same row, if possible. When deleting the last record, the record pointer will move to the previous record. Delete triggers the following events: Validate Reposition Delete

Method	Parameters	Description
FieldCount		Returns the number of fields in *TableName*.
Insert		By default, adds a new record to the table, closes the RecordSet, and clears any data aware fields. This behaviour can be overridden in the Insert event. Insert triggers the following events: Validate (and closes RecordSet) Insert (and clears data-aware fields)
MoveFirst		Moves to the first record in the RecordSet.
MoveLast		Moves to the last record in the RecordSet.
MoveNext		Moves to the next record in the RecordSet.
MovePrevious		Moves to the previous record in the RecordSet.
MoveTo	Index as Integer	Moves to the *Index* row in the RecordSet.
RecordCount		Returns the number of records in *TableName*.
RecordSet		Returns a RecordSet.
Row		Returns the current row in the RecordSet.
RowCount		Returns the number of rows in *TableName*. Obsolescent; use RecordCount instead.
RunQuery		Repopulates the RecordSet by running *SQLQuery*.
Update		Updates the current record in the selection, leaving the RecordSet at the same row. Update triggers the following events: Validate Update

Notes

When you add a DataControl to a window, it looks like this:



You use a DataControl by creating a window that uses EditFields, ListBoxes, Popup menus, or StaticText controls to display and edit data. Each of these controls has two properties that are meaningful only when used in conjunction with a DataControl. A primary key field is a required field whose values are unique. The easiest way to generate data for a primary key is to use sequential numbering. See the example below.

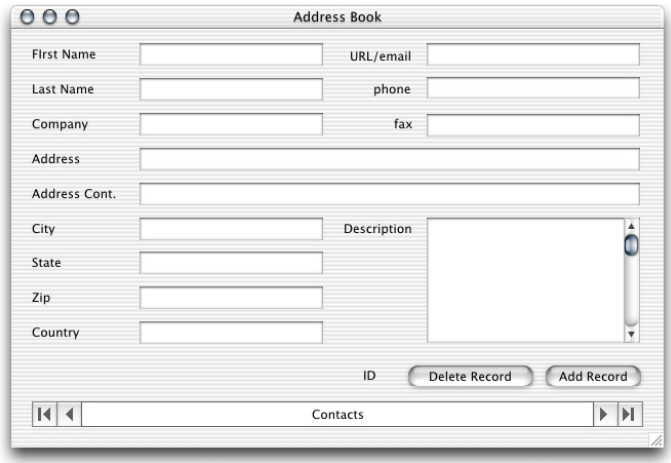
You can use Object Binding to bind objects such as a PushButton to a DataControl. Supported actions are: Add, Update, and Delete Record when the PushButton is

pushed. Shift-Command drag from the control to the **DataControl** to set an action with Object Binding.

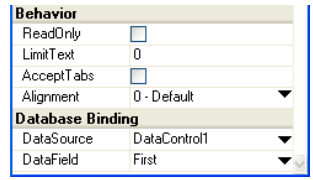
When you build an interface using a **DataControl**, it makes it easy to modify and save records. When the user saves changes, it automatically updates the values in the database back end with the values in the controls bound to the DataControl. If you need to update other values, you need to do it conventionally using the RecordSet method of the DataControl.

Example

The following example database displays fields in EditFields and uses a **DataControl** for record navigation.



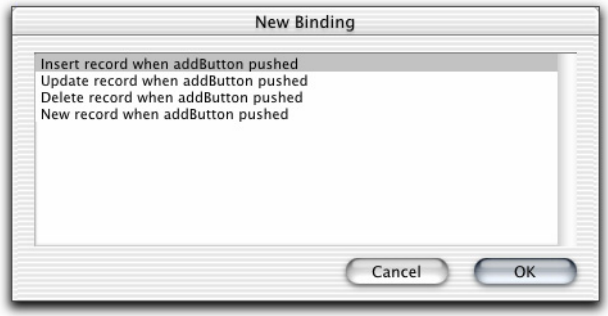
Each EditField is linked to the **DataControl** in the Properties window by setting the DataSource and DataField properties. When you set the DataSource, a pop-up menu of fields from the table specified by Database becomes available for the DataField property. The following illustration shows that the EditField for First Name is bound to the column "first" in the table to which the **DataControl** is linked.



The Primary Key field, ID, is linked to an EditField named "IDField" in this manner.

The Add Record button is linked to the **DataControl** using object binding. To establish the bind, Shift-Control drag from the PushButton control to the

DataControl to display the New Binding dialog box and select "Insert Record when addButton pushed".



Use the same process to bind the Delete button to the DataControl and use "Delete Record when DeleteButton pushed."

In the Validate event for the **DataControl**, use this code to increment the primary key:

```
Dim dbc as RecordSet
If action = 0 then
 dbc = Me.database.SQLSelect("Select Max(id) from " + me.tableName)
 idfield.text = Str(dbc.idxField(1).integerValue + 1)
end if
```

See Also

Database, DatabaseCursor, DatabaseRecord, RecordSet classes.

Datagram Class

Used to send and receive data using the UDPSocket class.

Super Class

Object

Properties

Name	Type	Description
Address	String	The address of the remote machine that you are sending data to, or receiving data from.
Data	String	The data you are sending or receiving.

Notes

A **Datagram** consists of two parts, the IP address of the remote machine which sent you the data, and the data itself. When you attempt to send data, you must specify information in the form of a Datagram. This information is usually the remote address of the machine you want to receive your packet, the port it should be sent to, and the

data you wish to send the remote machine. Please see the UDPSocket for information on how to use the **Datagram.**

See Also

UDPSocket, SocketCore classes.

DataNotificationReceiver Interface Class

Used to program custom object bindings.

Methods

Name	Parameters	Description
Reload		Reloads data notifier after changes
Commit		Commits changes to data.

DataNotifier Interface Class

Used to program custom object bindings.

Methods

Name	Parameters	Description
addDataNotificationReceiver	receiver as DataNotificationReceiver	Adds a data notification receiver to the interface class
removeDataNotificationReceiver	receiver as DataNotificationReceiver	Removes a data notification receiver from the interface class.

Date Class

A **Date** value stores the number of seconds since January 1, 1904. Properties of a **Date** enable you to get and set a day value only, a date/time, or only a time.

Super Class

Object

Properties

Name	Type	Description
AbbreviatedDate	String	Example: Wed, Dec. 31, 1997
Day	Integer	The day number of the date.

Name	Type	Description
DayOfWeek	Integer	1=Sunday, 7=Saturday
DayOfYear	Integer	The number days into the year that the date falls on.
Hour	Integer	The hour number of the time portion of the date.
LongDate	String	Example: Wednesday, December 31, 1997
LongTime	String	Example: 2:32:40 PM
Minute	Integer	The minute number of the time portion of the date.
Month	Integer	The month number of the date.
Second	Integer	The second number of the time portion of the date.
ShortDate	String	Example: 12/31/97
ShortTime	String	Example: 2:32 PM
TotalSeconds	Double	The number of seconds since January 1, 1904. TotalSeconds is the 'master' property from which other ways of expressing date/time are derived. A negative value of TotalSeconds indicates a Date/Time prior to January 1, 1904.
WeekOfYear	Integer	The number of the week of the year the date falls in.
Year	Integer	The year portion of the date.

Notes

When you create a **Date**, it doesn't actually get the values of the current date and time until you access one of its properties. This initializes the object.

The date properties of FolderItems can be accessed via the CreationDate and ModificationDate properties of FolderItem objects. You can get the current date and time by creating a new date and reading the values of the Year, Month, Day, Hour, Minute, and Second properties.

In the following code:

```
Dim v as Variant
Dim d as Date
d=New Date
v=d
```

What is actually happening is that the Variant stores the value of the TotalSeconds property as a Double, along with the type information that it is a **Date** (the Variant's Type property = 7).

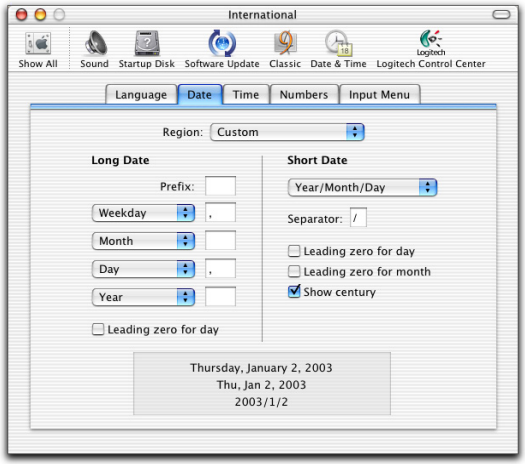
Although **Date** is a class that is subclassed from Object, VarType identifies a **Date** as a Date data type (Type=7) rather than an Object (Type=9).

Use the ParseDate function to convert a date string to a **Date** value.

The TotalSeconds property is the 'master' property that stores the date/time associated with a **Date**. The other property values are derived from TotalSeconds. If you change

the value of the TotalSeconds property, the values of the Year, Month, Day, Hour, Minute, and Second properties change to reflect the second on which TotalSeconds occurs. Conversely, if you change any of these properties, the value of TotalSeconds changes commensurately.

The **Date** properties that return formatted date or time information are affected by the settings in the user's Date and Time Control Panel. For example, the Date Format settings shown below produce the following output when the Date class example code is run:



If you need to control the exact appearance of date/time information, the best way is to extract the information yourself and manage the formatting using string manipulation functions.

Examples

This example creates a **Date** and displays the current date in a message box.

```
Dim d as Date
d=New Date
MsgBox d.shortdate
```

See Also

ParseDate, Ticks, Microseconds functions; FolderItem class.

DebugBuild Constant

Used to determine the current operating environment.

Syntax

***result*=DebugBuild**

Part	Type	Description
result	**Boolean**	True if the application is running within the IDE, i.e., you chose Debug ▶ Run.

Example

The following example displays a message box only in the IDE.

```
#if DebugBuild then
  MsgBox "You're in the IDE"
#endif
```

See Also

TargetMacOS, Target68K, TargetPPC, TargetCarbon, TargetWin32, RBVersion, RBVersionString constants; #If statement.

DebugDumpObjects Method

Used to determine if your application is leaking memory.

Syntax

DebugDumpObjects (*filename*)

Part	Type	Description
filename	String	The name of the text file that will be created when the method is called.

Notes

REALbasic manages memory for you. This means that it allocates memory for new objects your application creates, keeps track of which objects your application is currently using and removes objects from memory when they are no longer in use. There have been situations where REALbasic was not removing objects from memory causing your application to use more and more memory the longer it's running. Your code can also create these situations. See, for example, the discussion of circular references in StackOverflowException.

The DebugDumpObjects method can help determine if you have a memory leak. When you call DebugDumpObjects, the method creates a text file in the same location as your application or REALbasic (if you are running your application in the IDE). This text file contains the amount of memory currently being used by objects in memory like windows, controls, instances of classes, etc. The text file also lists all of the objects in memory that are using memory. You can determine if a memory leak exists by

calling this method at various times and then comparing the text files that are created to determine if the objects in memory are what you are expecting to be in memory.

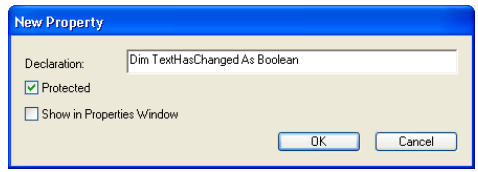
Example The following line of code creates the text file described above:

```
DebugDumpObjects ("dumpfile")
```

Declaration: Contains a Reserved Word Error

You used a Dim statement while declaring a property in the New Property dialog box.

Example The Dim keyword should be omitted from the declaration, "Dim TextHasChanged as Boolean":



Declare Statement

Used to make toolbox calls. PPC and X86 machines are currently supported. REALbasic versions 3.5.2 and earlier support 68K machines as well. DLLs can be called from a built Windows application.

Syntax **Declare Sub/Function *Name* Lib *Fragment* [Alias *AliasName*] ([*Parameters*]) [As *ReturnType*] [Inline68K *EntryPoint*]**

Part	Type	Description
Name	Name	Name of toolbox call. Use *Name* in your code to refer to the toolbox call.
Fragment	String	Library in which toolbox call is found.
AliasName	String	Optional: If the toolbox call has the same name as a REALbasic method, declare the call with a different name and use the alias to refer to the actual toolbox call.
Parameters		Parameters of toolbox call.
ReturnType		Optional: The data type of the value returned by the call if it is a function.
EntryPoint	String	Optional: If you want to include 68K Mac support, include the InLine68K parameter.

Notes

Function names for the OS (Mac or Win32) are case sensitive. For example, SetLocalTime works on Win32 but SetlocalTime does not.

Declare statements must call CarbonLib to run on Mac OS X. If they do not, the application will run in the "classic" environment within Mac OS X.

A value passed to a **Declare** statement must be a constant or a local variable. It cannot be the result of a function call or a property. To pass a value, simple declare a variable and set the property of function result equal to the variable.

The **Declare** statement allows use of constants for Lib names.

Examples

The following example uses the Macintosh Speech Manager to speak the text in an EditField. Note that conditional compilation is used both for the **Declare** statement and the toolbox call itself. In general, if the functionality is available on both platforms, you should code for both using the same Name and then use TargetBoolean constants to call the correct version for the current platform. In general, the Windows version of a toolbox call will take different parameters than the Mac version.

```
Dim s as String
Dim i as Integer
#If TargetMacOS
 Declare Function SpeakString lib  "SpeechLib" (SpeakString as pstring) as
 Integer
#Endif

s=EditField1.text
#If TargetMacOS
 i=SpeakString(s)
#Else
 MsgBox "Speech is supported only on Macs!"
#Endif
```

68K Macintosh

If you are using a version of REALbasic that supports 68K hardware, you can add the Inline68K keyword to the Declare statement, as follows:

```
Declare Function SpeakString lib "SpeechLib" (SpeakString as pstring) as
Integer Inline68k("203C0220000CA800")
```

Note: The final version of REALbasic that built 68K applications was version 3.5.2.

Name Conflicts

If SpeakString were also the name of a REALbasic method, you would have to use the Alias keyword to refer to the toolbox call. For example, the following statement works:

```
Declare Function MySpeakString lib "SpeechLib" Alias "SpeakString"
(SpeakString as pstring)  as Integer
.
.
i=MySpeakString(s)
```

The following example displays the maximum number of ticks between mouse clicks for two clicks to be considered a "double-click." Above this value, two clicks are considered separate mouse events. On the Mac, the time can be adjusted in the Mouse Control panel.

```
Dim DoubleClickTime as Integer
#if TargetMacOS
Declare Function GetDblTime Lib "InterfaceLib" () as Integer Inline68K("2EB802F0")
doubleClickTime = GetDblTime()
#endif

#if TargetWin32
Declare Function GetDoubleClickTime Lib "User32.DLL" () as Integer
doubleClickTime = GetDoubleClickTime()
#endif
MsgBox Str(doubleclicktime)
```

To modify the code to run under Carbon, substitute "CarbonLib" for "InterfaceLib" and delete the InLine68K reference:

```
Dim DoubleClickTime as Integer
#if TargetCarbon
Declare Function GetDblTime Lib "CarbonLib" () as Integer
doubleClickTime = GetDblTime()
#endif
MsgBox Str(doubleclicktime)
```

See Also

TargetMacOS, Target68K, TargetPPC, TargetCarbon, TargetWin32 constants.

DecodeBase64 Function

Decodes a Base64 string back to its original form.

Syntax

***result*=DecodeBase64(*str*)**

Part	Type	Description
result	String	The decoded value of *str*.
str	String	The Base64 String to be decoded.

Notes

The **DecodeBase64** function performs the reverse operation of EncodeBase64. An encoded string passed to **DecodeBase64** returns the original string.

See Also

EncodeBase64 function.

DecodeQuotedPrintable Function

Decodes into 'normal' text a string that has been encoded by EncodeQuotedPrintable.

Syntax

***result*=DecodeQuotedPrintable(*str*)**

Part	Type	Description
result	String	The result of processing *str*.
char	String	The string expression to be decoded back to its original form.

Notes

DecodeQuotedPrintable converts text representations of unprintable characters (i.e., control characters, returns and tabs) back into their original form. It undoes the encoding done by EncodeQuotedPrintable.

The **DecodeQuotedPrintable** function performs the function in reverse.

See Also

EncodeQuotedPrintable function.

DefineEncoding Function

Returns a String with the same data as the given string, but with the encoding of the given encoding. This function is useful when you have a string whose encoding is known to you but not to REALbasic.

Syntax

***result*=DefineEncoding(*str*, *enc*)**

Part	Type	Description
result	String	The result of encoding *str* using the encoding specified by *enc*.
str	String	The String to be encoded.
enc	TextEncoding	The TextEncoding to be used to encode str.

Notes

Consult the values of *Base* in Appendix A of the printed or pdf version of the Language Reference when creating the TextEncoding object using the GetTextEncoding function.

Example

The following example encodes the text in an EditField according to the value of the passed TextEncoding object.

```
Editfield2.text=DefineEncoding(EditField1.text,GetTextEncoding(33))
```

See Also

TextEncoding class; GetTextEncoding function.

DesktopFolder Function

Used to access items on the desktop. On Windows, it accesses the Desktop directory.

Syntax

***result*=DesktopFolder**

Part	Type	Description
result	FolderItem	A FolderItem that represents the items on the desktop.

Notes

Use the **DesktopFolder** function to access items on the desktop. Though they don't appear to be in a folder, in fact, all items on the desktop are in a folder called DesktopFolder.

Volumes and the Trash Can or Recycle Bin are not included in the Desktop Folder; however, desktop printers are. The Trash Can can be accessed via the TrashFolder function.

The **DesktopFolder** function provides a way to access the Desktop folder that will work under different language systems, and will continue to work even if Apple reorganizes the System folder. If your application will run on Windows, you can check the TargetWin32 global constant before calling **DesktopFolder**.

Examples

See the AppleMenuFolder function for an example.

See Also

AppleMenuFolder, ApplicationSupportFolder, ControlPanelsFolder, ExtensionsFolder, FontsFolder, PreferencesFolder, StartupItemsFolder, SystemFolder, TemporaryFolder, TrashFolder functions.

Destructors Can't Have Parameters Error

You cannot define parameters for the destructor for a class. The Method declaration dialog box does not allow this.

A Destructor has the same name as the class, preceded by the '~' sign. It executes automatically when the class goes out of scope.

Dictionary Class

An object that contains a list of *key-value* pairs. Both keys and values are Variants.

Super Class

Object

Properties

Name	Type	Description
BinCount	Integer	The number of bins the hash table uses. This is a measure of the hash table size, independent of the number of items the Dictionary contains. Normally, you do not need to be concerned with bins and the hash table. BinCount would be of use to you only if you have very large dictionaries consisting of thousands of entries and you want to try to optimize performance. If you set BinCount to a positive value, the Dictionary will use that number of bins regardless of the number of items in the Dictionary. If you set BinCount to a value equal to or less than zero, the Dictionary will pick whatever value it thinks is appropriate for the number of items in the dictionary. It will dynamically change the number of bins as items are added or removed from the dictionary (or until you assign a positive value to BinCount). The latter is the default behaviour.

Name	Type	Description
Count	Integer	Number of key-value pairs in the Dictionary.
Value	Variant	Takes one parameter, key as Variant. Assigns a value to the *key* item in the Dictionary or retrieves the value associated with the *key* item. If you attempt to read a value for a key that is not in the dictionary, a KeyNotFoundException error is raised.

Methods

Name	Parameters	Description
Clear		Clears all the key-value pairs in the Dictionary.
HasKey	Key as Variant	Returns a Boolean. Returns True if *Key* is in the Dictionary and False if it is not.
Key	Index as Integer	Returns a Variant. Returns the value of *key* for the *Index*th sequential item in the Dictionary. The first item is numbered zero. If there is no *Index*th item in the Dictionary, a call generates an OutOfBoundsException error.
Remove	Key as Variant	Removes the *key* value-key pair from the Dictionary. If *Key* is not in the Dictionary, it raises a KeyNotFoundException error.

Notes

The **Dictionary** class provides the functionality of the Collection class and offers several advantages: With the Collection class, the time taken to locate an item is a function of the number of items in the Collection because the search is sequential. A Dictionary uses a hash table, making the time (relatively) independent of the number of items. It is designed for high-speed lookups. Also, the key parameter in a **Dictionary** is a Variant, but is a String in the Collection class, giving you greater flexibility. On the other hand, the Collection can store multiple values per key. When you assign a value to a key that is already in the Dictionary, the new value replaces the old one.

Example

The following For loop loads the values of a **Dictionary** into a ListBox.

```
Dim d as Dictionary
Dim i as Integer
d=New Dictionary
d.Value(1)="ID"
d.Value(2)="Lois Lane"
d.Value(3)="Reporter"
d.Value(4)=84000
For i=1 to d.Count
 ListBox1.Addrow d.value(i)
Next
```

The following example takes advantage of the fact that the key is a Variant, and not necessarily a number. In this example, the keys are colors and the values are descriptions.

```
Dim d as Dictionary
d=New Dictionary
d.Value(RGB(255,0,0))="This is pure red."
d.Value(RGB(0,255,255))="This is pure cyan."
MsgBox d.value(d.Key(1)) //retrieves cyan
Exception err as RuntimeException
 If err IsA KeyNotFoundException then
  MsgBox "Key not in the dictionary"
 End if
 If err IsA OutOfBoundsException then
  MsgBox "The index of the key is out of bounds!"
 End if
```

If the index passed to the Key method in the line:

```
MsgBox d.value(d.Key(1)) //retieves cyan
```

is not an element in the dictionary, an OutOfBoundsException occurs and the Exception block at the end of the method will trap it. You could also retrieve this value in the dictionary by passing the Value method the key rather than the key's index:

```
MsgBox dict.value(RGB(0,255,255))
```

In this case, if you pass a key that is not in the dictionary, a KeyNotFoundException will occur and the Exception block will also trap it and display the appropriate error message.

Instead of the Exception block, you could first use the HasKey method before trying to access the value:

```
If d.HasKey(RGB(0,255,255)) then
 MsgBox d.value(RGB(0,255,255))
Else
 MsgBox "Key not in the dictionary!"
End If
```

See Also

KeyNotFoundException error; Variant, Collection classes.

Dim Statement

Creates a local variable with the name and size (in the case of an array) and data type specified.

Syntax

The **Dim** statement has two syntaxes:

Dim *variableName* [,*variableNameN*] As *dataType*

Part	Type	Description
variableName	Variable name	The name of the new variable.
variableNameN	Variable name	Optional. The names of other variables you wish to create with the same data type. Each name should be separated with a comma.
dataType	Data type name	The data type of the variable.

Dim *arrayName*(*size* [,*sizeN*]) as *dataType*

Part	Type	Description
arrayName	Variable name	The name of the new array.
size	Integer	The size (number of elements) for the array.
sizeN	Integer	Optional. The size of the next dimension of the array if you are creating a multi-dimensional array.
dataType	Data type name	The data type of the array.

Notes

In the first syntax, the **Dim** statement is used to create new variables. A variable is an object stored in RAM that can hold a value. In the second syntax, the **Dim** statement is used to create a new array of the size specified. An array is simply a variable that can contain multiple values that are all the same data type. Passing more than one size creates a multi-dimensional array.

The maximum number of dimensions in REALbasic is 20.

The **Dim** statement cannot be placed inside a conditional structure (an If or Case statement) or a loop structure, such as While and For. In general, it *can* be placed after other lines of executable code.

A Collection is similar to an array, except that the elements of a collection can be of different data types.

Variables created with the **Dim** statement are local in scope. This means that they are created each time the method is called and are destroyed when the method is finished. The value of a local variable can only be accessed from within the method.

Properties created within a module, however, are global in scope and can be accessed throughout the application.

If you don't know the size of the array you need at the time you declare it, you can declare it as a null array, i.e., an array with no elements, and use the Redim command to resize it later. You do this by giving it an index of -1. This means "an array of no elements." For example, the statement:

```
Dim aNames (-1) as String
```

creates the array aNames with no elements. If your program needs to load a list of names that the user enters, you can wait to size the array until you know how many names the user has entered. You can also accomplish this by leaving out the -1. The following statement is equivalent:

```
Dim aNames () as String
```

Examples

This example uses the **Dim** statement to create an integer variable called "age" and assigns it the value 33.

```
Dim age As Integer
age=33
```

This example uses the **Dim** statement to create two string variables.

```
Dim FirstName, LastName As String
```

This example uses the **Dim** statement to create an array called aNames with 11 elements (remember, arrays have a zero element).

```
Dim aNames(10) As String
```

This example uses the **Dim** statement to create an array called aNames with one element.

```
Dim aNames(0) As String
```

This example uses the **Dim** statement to create a two-dimensional array called aNames with 11 rows and 4 columns.

```
Dim aNames(10,3) As String
```

See Also

Append, Insert, Redim, Remove, Sort methods; UBound function; Collection class.

DisclosureTriangle Control 

Used to add a disclosure triangle to a window. A DisclosureTriangle can face either left or right in its "closed" position. When the user clicks a DisclosureTriangle control, the value of *Value* is toggled automatically.

Super Class

RectControl

Because this is a RectControl, see the RectControl for other properties and events that are common to all RectControl objects.

Properties

Name	Type	Description
Facing	Integer	0 - Right facing 1 - Left facing
Value	Boolean	True corresponds to downward; False corresponds to either left or right depending on the value of *Facing*.

See Also

RectControl class.

DocumentsFolder Function

Used to access the Documents folder. On Windows XP, a call to DocumentsFolder returns the My Documents folder in the current user's directory, e.g., C:\Documents and Settings*username*\My Documents.

Syntax

***result*=DocumentsFolder**

Part	Type	Description
result	FolderItem	A FolderItem that represents the Documents folder.

Notes

Use the **DocumentsFolder** function to access the user's Documents folder.

The **DocumentsFolder** function provides a way to access the Documents folder that will work under different language systems and on both Macintosh and Windows.

Examples

See the AppleMenuFolder function for an example.

See Also DesktopFolder, ExtensionsFolder, FontsFolder, ApplicationSupportFolder, PreferencesFolder, ShutDownItemsFolder, StartupItemsFolder, SystemFolder, TemporaryFolder, TrashFolder functions.

Do...Loop Statement

Repeatedly executes a series of statements while a specified condition is True.

Syntax

Do [Until *condition*]

[*Statements*]

[Exit]

[*Statements*]

Loop [Until *condition*]

Part	Description
Do [Until]	Begins the loop. If the optional **Until** and *condition* are included, the *condition* is evaluated to determine if the loop should exit.
Condition	Any valid Boolean expression. When this expression evaluates to True, the loop will exit.
Exit	Optional. Forces the loop to exit prematurely.
statements	Statements to be executed repeatedly (until *condition* evaluates to False).
Loop[Until}	Ends the loop. If the optional **Until** and *condition* are included, the *condition* is evaluated to determine if the loop should exit.

Notes

The **Do...Loop** statement continues to execute *statements* repeatedly until *condition* is True or the Exit statement is executed.

If *statements* should only be executed if *condition* is already True, test for *condition* at the beginning of the loop. If *statements* should execute at least once, test *condition* at the end of the loop.

If *condition* is not used to cause the loop to exit, your logic must call the Exit statement somewhere inside the **Do...Loop** or the loop will never end.

Do...Loop statements can be nested to any level. Each **Loop** statement goes with the previous **Do** statement.

When a loop runs it takes over the interface, preventing the user from interacting with menus and controls. Ordinarily, this is of no concern. If, however, the loop is very lengthy, you can move the code for the loop to a separate Thread, allowing it to execute in the background.

Example

This example uses the **Do...Loop** statement to increment a variable. If the variable is greater than 100 before the loop begins, the variable will not be incremented.

```
Do Until x<100
  x=x+1
Loop
```

In this example, the variable will be incremented at least once because *condition* is tested at the end of the loop.

```
Do
  x=x+1
Loop Until x<100
```

See Also

For...Next, While...Wend statements; Thread class.

Double Data Type

A **Double** is an intrinsic data type in REALbasic. A **Double** is a number that can contain a decimal value, i.e., a real number. It can take on a value between 2.2250738585072013 e-308 and 1.7976931348623157 e+308. In other languages, REALbasic's **Double** may be referred to as a double precision real number. Because **Doubles** are numbers, you can perform mathematical calculations on them. **Doubles** use 8 bytes of memory.

The VarType function returns a value of 5 when passed a **Double**.

Example

The Single and **Double** data types allow you to store and manage floating point numbers.

```
Dim d as Double
d=3.14159265358979323846264338327950
```

See Also

Integer, Single, String, Boolean data types; -, +, *, /, <, <=, =, >=, >, <>, \, Mod, Val, Str, VarType functions.

DragItem Class

Used to transfer data being dragged by the user.

Super Class Object

Properties

Name	Type	Description
Destination	Object	The location of the dropped item when you drop it to the desktop or onto an application. Since *Destination* is of type Object, it potentially can return any type of REALbasic object. Currently, *Destination* is set up only to work with the Macintosh Finder and will only return a FolderItem. It requires either Mac OS 8-9 or Mac OS X 10.2 or above. To ensure that your application is compatible with future versions of REALbasic, check the type of object it returns with IsA before recasting it to a FolderItem. On Windows, dragging items to the Desktop is not supported; therefore this property does not work under that condition.
DropHeight	Integer	Height of object being dropped.
DropLeft	Integer	Distance from the left side of control where the object was dropped.
DropTop	Integer	Distance from the top of the control where the object was dropped.
DropWidth	Integer	Width of object being dropped.
FolderItem	FolderItem	The FolderItem being dragged.
FolderItemAvailable	Boolean	The *FolderItem* property contains a FolderItem.
MacData	String	The file Type of the data being dragged. Type is a four-character resource type or programmer-defined four-character code. Supported only on Macintosh and within the REALbasic application on Windows.
MacDataAvailable	Boolean	True if the DragItem contains data of the type specified. Type is a four-character resource type or programmer-defined four-character code. The type is passed to MacData or PrivateMacData as a string. Supported only on Macintosh.
MouseCursor	MouseCursor	The cursor to be displayed during the drag.
Picture	Picture	The Picture being dragged.
PictureAvailable	Boolean	The *Picture* property contains a picture.

Name	Type	Description
PrivateMacData	String	The data (of the Type specified) being dragged. Type is a four-character resource type or programmer-defined four-character code. This data cannot be dragged to another application. Supported only on Macintosh.
Text	Text	The text being dragged.
TextAvailable	Boolean	The Text property contains text.

Methods

Name	Parameters	Description
AddItem	Left as Integer, Top as Integer, Width as Integer, Height as Integer	Adds an item to the drag object. The parameters specify the drag rectangle to be displayed.
Drag		Allows the drag to take place.
NextItem		Returns True if there is another item that has been dragged and assigns the next item's values to the DragItem properties.

Notes

When the user wishes to drag some data from a window or control, a new **DragItem** object must be created to hold that data in order to transfer it to the object the data will be dropped on. REALbasic implements true Macintosh drag and drop — which means that the recipient of the data can be anything that supports drag and drop and supports the kind of data being dragged. For example, text can be dragged to the desktop to create a text clipping file or any application that accepts dropped text (like SimpleText or the NotePad). Pictures can be dragged to the Desktop to create picture clipping files. Other types of data can be dragged using the MacData property.

By default, a **DragItem** can contain only one row containing a single entry for each data type. Using the AddItem method, you can add more rows to the **DragItem** allowing multiple entries of the same data type. For example, if you wanted a drag item to contain two separate pictures, you would assign one picture to the Picture property then call the AddItem method to add a row and then assign the second picture to the Picture property.

The MacData and PrivateMacData allow you to support drag and drop for other data formats. You specify the data format using a four-character Type, corresponding to a Macintosh resource type. MacData and PrivateMacData return the dragged item as a string. It is up to the control or window that receives the **DragItem** to manage it. Typically, this will mean that you will have to make toolbox calls or use a plug-in that is designed to handle the data format.

You can also use MacData or PrivateMacData to control internal drag and drop. If you make up a four-character Type (not an existing Macintosh resource type), you can prevent a control from receiving dragged text from other sources. See the example of dragging from one ListBox to another.

For EditFields, drag and drop is supported only if the MultiLine property is True (checked). There is no need to create a **DragItem** for EditFields as this is handled automatically by REALbasic.

ListBoxes support the dragging of rows automatically if the EnableDrag property of the ListBox is True (checked). Like EditFields, REALbasic creates the **DragItem** for the ListBox automatically but you must populate the **DragItem** in the DragRow event handler. See "ListBox Control" on page 247 for more information on dragging from ListBoxes.

The DropWidth and DropHeight properties have no meaning in Win32 and will return 0 unless set using NewDragItem.

Examples

This example shows how to implement dragging of the picture in a Canvas control. This example uses the Me function to refer to the Canvas control. Just for fun, the example uses a custom cursor for the drag. The example assumes that a resource file named Resources containing a CURS resource with ID 1044 has been dragged into the Project window.

```
Function MouseDown(X As Integer, Y As Integer) As Boolean
 Dim d as DragItem
 d=NewDragItem(Me.left,Me.top,Me.width,Me.height)
 d.picture=Me.Backdrop
 d.MouseCursor=app.ResourceFork.getcursor(1044)
 d.Drag //Allow the drag
End Function
```

See also the example for ImageWell, which illustrates drag and drop between two ImageWells and PICT files or clippings on the desktop and the ListBox control, which has an example of drag and drop between two ListBoxes.

The following example implements a 'mover' interface in which a user can drag rows from ListBox1 to ListBox2 (and in the reverse direction), but neither ListBox accepts dragged text from other controls or from outside the REALbasic application. It uses PrivateMacData since it also prevents text from being dragged outside the application.

To permit drag and drop in both directions, the two ListBoxes are set up in an identical manner.

Each ListBox has its EnableDrag property set to True. The DragRow event handler is:

```
Function DragRow (drag as DragItem, row as Integer) As Boolean
 Drag.PrivateMacData("hehe")=Me.List(Row)+Chr(13) //get the text
  Me.RemoveRow(Row)
 Return True //allow the drag
End function
```

To enable each ListBox to accept the dragged text, the following statement appears in its Open event handler:

```
Me.AcceptMacDataDrop("hehe")
```

The DropObject event handler is as follows:

```
Sub DropObject (obj as DragItem)
 If  obj.MacDataAvailable("hehe") then
  Me.AddRow(obj.MacData("hehe"))
 end if
End Sub
```

The following example allows the user to drag several text files from desktop to an EditField. The example places the contents of all the text files in the EditField, appending each file's contents to the Text property of the EditField.

To support several files, the NextItem function is used to determine whether there are any more files remaining to be dropped.

```
Dim textStream as TextInputStream
If obj.folderItemAvailable then
  Do
    textStream=obj.FolderItem.Openastextfile
    Me.text=Me.text+textstream.readall+Chr(13)
  loop until not obj.NextItem
End if
```

The EditField has the line:

```
Me.AcceptFileDrop("text")
```

in its Open event handler, where "text" is defined in the File Types dialog as files of Macintosh type TEXT.

See Also

EditField, ListBox, Canvas, ImageWell controls; Picture, FolderItem classes.

EditableMovie Class

Used to create, manipulate, and save QuickTime movies.

Super Class

Movie

In previous versions of REALbasic, **EditableMovie** was subclassed from Object. Since it is now subclassed from Movie, you can view an **EditableMovie** using the MoviePlayer control.

Properties

Name	Type	Description
Duration	Double	Duration of the movie (seconds)
EOF	Boolean	True if the Position is at the end of the movie.
Handle	Integer	Returns a handle to the **EditableMovie**, mainly used for toolbox calls that require the handle to a movie.
Picture	Picture	Picture of the movie at the current position.
Position	Double	Current position of the movie (seconds).
Poster	Picture	Poster frame of the movie.
SaveOnClose	Boolean	If True, the **EditableMovie** will update and save the file when it goes out of scope (as in prior versions of REALbasic). If False, no updates to the **EditableMovie**'s file will occur unless you call the CommitChanges method.
SelLength	Double	The length (seconds) of the selected segment of the EditableMovie.
SelStart	Double	The starting position (seconds) of the selected segment of the EditableMovie
TimeDuration	Integer	Duration of the movie, in TimeScale units.
TimeScale	Integer	Defines the timescale of the movie, in 1/TimeScale units per second.
TimeValue	Integer	Current position of the movie, expressed in TimeScale units.
TrackCount	Integer	Number of tracks in the movie.
UserData	QTUserData	UserData contains additional information about the movie, including credits, copyright information, etc. The QTUserData class is used to manage the individual pieces of data.

Methods

Name	Parameters	Description
AppendMovieSegment	SourceMovie As Movie, SourcePosition As Double SourceDuration As Double, ShowProgress As Boolean	Appends *SourceMovie* to the end of the calling **EditableMovie**. *SourcePosition* is the beginning (in seconds) of the segment in SourceMovie to be appended to the calling movie. *SourceDuration* is the length (in seconds) in *SourceMovie* of the segment to be appended. If the operation is time-consuming, QuickTime will display a modal dialog with a progress indicator if *ShowProgress* is True.
Clear		Clear the **EditableMovie**'s current selection. Use the **EditableMovie**'s SelStart and SelLength properties to set the selection.
CommitChanges		Returns a Boolean. It returns True if the commit is successful. Commits changes to the **EditableMovie** manually. You can also do this using the class's destructor, as in previous versions of REAlbasic. And, if you set SaveOnClose to True, it will save changes automatically when the **EditableMovie** goes out of scope.
Copy		Copies the **EditableMovie**'s current selection to the Clipboard. Use the **EditableMovie**'s SelStart and SelLength properties to set the selection.
Cut		Cuts the **EditableMovie's** selection to the Clipboard. Use the **EditableMovie**'s SelStart and SelLength properties to set the selection.
DataSize	StartTime as Integer DurationTime as Integer	Returns an Integer, the number of bytes in the segment specified by *StartTime* and *DurationTime*.

Name	Parameters	Description
Flatten	Destination As FolderItem, ResourceFork As Boolean, FastStart as Boolean, ShowProgress As Boolean	Flattens the EditableMovie and saves it in *Destination*. Macintosh users can choose to flatten the ResourceFork by setting ResourceFork to True; Windows users will always flatten the data fork. This parameter is ignored in Windows builds. Set *FastStart* to True to enable QuickTime's "FastStart" feature for internet streaming. If the operation is time-consuming, QuickTime will display a modal dialog with a progress indicator if *ShowProgress* is True.
InsertMovieSegment	sourceMovie As Movie, sourcePosition As Double, sourceDuration As Double, destinationPosition As Double, showProgress As Boolean	Inserts a segment from *SourceMovie* at the position specified by *DestinationPosition*. The segment in SourceMovie to be inserted begins at *SourcePosition* and continuing by *SourceDuration*. *SourcePosition*, *SourceDuration*, and *DestinationPosition* are in seconds. If the operation is time-consuming, QuickTime will display a modal dialog with a progress indicator if *ShowProgress* is True.
NewSoundTrack		Creates and returns a QTSoundTrack. You cannot create a new QTSoundTrack with the New function.
NewVideoTrack	Width as Integer Height as Integer TimeScale as Integer	*Width* and *Height* are in pixels. *TimeScale* is in frames per second. Creates a new video track and returns a QTVideoTrack.
Paste		Pastes the contents of the Clipboard (from calling Cut or Copy) to the calling **EditableMovie** at the SelStart position. Use the SelStart and SelLength properties to set the selection. If SelLength is greater than 0, the contents of the Clipboard will overwrite the selected contents of the **EditableMovie**.
RestoreRedoEditState		Restores the **EditableMovie** to the state it was in when SetRedoEditState was called.
RestoreUndoEditState		Restores the **EditableMovie** to the state it was in when SetUndoEditState was called.

Name	Parameters	Description
ScaleMovieSegment	StartPosition As Double, OldDuration As Double, NewDuration As Double, ShowProgress As Boolean	Change the duration of the selected segment in the **EditableMovie**. Note: A SelStart of 0 and a SelLength of myEditableMovie.Duration can be used to scale the entire movie. A change in duration will result in a pitch change in the movie as well. For example, making the movie longer causes the pitch to be lowered. StartPosition is the beginning of the segment to be scaled. OldDuration is the current duration of the segment and NewDuration is the duration the segment should be scaled to. If the operation is time-consuming, QuickTime will display a modal dialog with a progress indicator if *ShowProgress* is True.
SetRedoEditState		Use this method to store an edit state for the **EditableMovie**. The **EditableMovie** can be returned to this state at anytime by calling RestoreRedoEditState, provided the EditableMovie hasn't gone out of scope.
SetUndoEditState		Use this method to store an edit state for the EditableMovie. The EditableMovie can be returned to this state at anytime by calling EditableMovie.RestoreUndoEditState provided the EditableMovie hasn't gone out of scope.
SoundTrack	Index as Integer	Retrieves the QTSoundTrack corresponding to the index that was passed. Index is 1-based array. Returns a QTSoundTrack.
VideoTrack	Index as Integer	Retrieves the video track corresponding to the index that was passed. Index is 1-based array. Returns a QTVideoTrack.
Track	Index as Integer	Index is 1-based array. Returns a QTTrack.

Notes

The *TimeScale* parameter defines the number of time units that pass each second. For example, a TimeScale value of 60 sets the timescale in sixtieths of a second. The *TimeValue* and *TimeDuration* parameters are expressed in TimeScale units.

The Position or the TimeValue properties can be used to position the movie; at that point, the Picture property contains the image at that point in the movie.

Examples

The following Pushbutton Action opens an existing movie as an **EditableMovie**. It updates the ImageWell and EditField with the current picture and position of the movie.

```
Dim f as FolderItem
Dim m as EditableMovie
f=GetFolderItem("Honey.Mov")
m=f.OpenEditableMovie
m.position=movieplayer1.position
ImageWell1.image=m.picture
Imagewell2.image=m.poster
EditField1.text=str(m.position)
```

The following example opens an existing movie as an **EditableMovie** and displays it in a MoviePlayer control named ThePlayer.

```
Dim f As FolderItem
Dim theEMovie as EditableMovie
f=GetOpenFolderItem("video/quicktime")
If f<>Nil and f.exists then
 theEMovie=f.OpenEditableMovie
 If theEMovie<>Nil then
  ThePlayer.movie=theEMovie
 end if
end if
```

The following example creates a new video track from two pictures, p1 and p2, that have been dragged into the Project window.

```
Dim sequence as QTEffectSequence
Dim i,effectID as Integer
Dim theEffect as QTEffect
Dim track as QTVideoTrack
Dim m as EditableMovie
Dim f as FolderItem
f=GetFolderItem("Movie")
m=f.CreateMovie
If radiobutton1.value then
 theEffect=GetQTCrossFadeEffect
else
 theEffect=GetQTSMPTEeffect(val(EffectsList.cell(EffectsList.ListIndex,1)))
end if
sequence=New QTEffectSequence(theEffect,p1,p2,96)
track=m.NewVideoTrack(p1.width, p1.height, 32)
```

See Also

QTTrack, QTUserData, QTVideoTrack, FolderItem classes.

EditField Control

The standard editable text field. An **EditField** control may contain one line of text or multiple lines of text, depending on the value of the MultiLine property. Text may either be of only one font, font size, and style or mixed fonts, font sizes, and styles ("Styled Text") depending on the value of the Styled property. Styled text can be printed using the StyledTextPrinter method.

EditFields, ListBoxes, and Canvas controls are the only controls that can have the focus on Mac OS.

Super Class

RectControl

Because this is a RectControl, see the RectControl for other properties and events that are common to all RectControl objects.

Properties

Name	Type	Description
AcceptTabs	Boolean	If True, pressing the Tab key will enter a tab into the EditField instead of moving the focus to the next item in the tab order.

Name	Type	Description
Alignment	Integer	Sets paragraph alignment for entire contents of EditField. 0: Default alignment 1: Left aligned 2: Centered 3: Right aligned Use SelAlignment to set paragraph alignments for individual paragraphs. Currently the Default alignment is the same as Left aligned.
BackColor	Color	The color of the background (white by default).
Bold	Boolean	Applies the bold style to the text.
Border	Boolean	If True, draws a border around the object.
DataField	String	Relevant only if the EditField is used in conjunction with a DataControl to display the contents of fields in a database table. Name of a field (String) in the table referenced by DataControl.
DataSource	DataControl	The DataControl that references a table in a database whose fields are displayed using EditFields (*TableName* as String). If assigned in code, you use the Name property of a DataControl. Use the DataField property to link each field to be displayed to an EditField. In the IDE, a popup menu of field names will appear.
Italic	Boolean	Applies the italic style to the text.
LimitText	Integer	The maximum number of characters allowed in the EditField. The value of zero does not limit text. Works only if MultiLine is not selected.
LiveUpdate	Boolean	If True, then the EditField will dynamically update the bound data value. Relevant when the EditField is bound to another control.
MultiLine	Boolean	If True, wrap text to multiple lines and allow more than 32K of text on all platforms. Page Up, Page Down, Home and End keys are supported for Multiline EditFields. A vertical scroll bar is added automatically when MultiLine is True. MultiLine must be set to True to support styled text and automatic drag and drop. MultiLine EditFields support any standard terminator (Return, Linefeed, or Return + Linefeed) to indicate the end of a paragraph. If MultiLine is off, EditFields have a limit of 32K of text on Mac OS "classic" builds.
Password	Boolean	If True, bullets appear in field instead of characters typed. The MultiLine property must be False to use the EditField as a Password field. The Cut and Copy Edit menu items are automatically disabled.
ReadOnly	Boolean	If True, the text cannot be modified.

Name	Type	Description
ScrollbarHorizontal	Boolean	If True, the EditField has a horizontal scrollbar and text does not wrap at the right edge of the EditField.
ScrollbarVertical	Boolean	If True, the EditField includes a vertical scrollbar. Called "Scrollbar" in earlier releases of REALbasic.
ScrollPosition	Integer	Zero-based index of the topmost visible line. Read ScrollPosition to determine the first visible line; write to ScrollPosition to scroll the EditField. Setting ScrollPosition to the last line or beyond the last line will simply vertical scrollbar's thumb to the bottom, not necessarily displaying the last line at the top of the EditField. ScrollPosition is 0 if the MultiLine property is False. Can be set in the IDE.
ScrollPositionX	Integer	Scrolls the EditField horizontally to a position X pixels from the left edge of the EditField.
SelAlignment	Integer	Controls paragraph alignment of selected text. 0: Default alignment 1: Left aligned 2: Centered 3: Right aligned -1: Mixed—Selection spans multiple paragraphs with different alignments. Currently, the Default alignment is the same as Left alignment.
SelBold	Boolean	Used to get and set the bold style of the highlighted text.
SelCondense	Boolean	Used to get and set the condensed style of the highlighted text. Not supported on Windows.
SelExtend	Boolean	Used to get and set the extended style of the highlighted text. Not supported on Windows.
SelItalic	Boolean	Used to get and set the italic style of the highlighted text.
SelLength	Integer	The number of highlighted characters. A SelLength of 0 means an insertion point rather than a selection. A value greater than the number of characters in the EditField means that the selection be from SelStart to the end of the EditField.
SelStart	Integer	The position of the first highlighted character. A value of zero means before the first character.
SelText	String	The currently highlighted text.
SelTextColor	Color	Used to get and set the text color of the highlighted text. When used in an unstyled EditField, it changes the color of all the text in the EditField.
SelTextFont	String	Used to get and set the text font of the highlighted text. When used in an unstyled EditField, it changes the font of all the text in the EditField.

Name	Type	Description
SelTextSize	Integer	Used to get and set the font size of the highlighted text. When used in an unstyled EditField it changes the font size of all the text in the EditField.
SelUnderline	Boolean	Used to get and set the underline style of the highlighted text.
Styled	Boolean	If True, styled text is allowed. The MultiLine property must also be set to True to permit styled text.
Text	String	The text displayed. Assigning a string to the Text property replaces the entire contents of the EditField. The font, size, and color are uniform and match the values last set with the TextFont, TextSize, and TextColor properties, or if these haven't been used, the settings in the IDE. If the EditField contains more than one fontsize, style, or font, you will lose these text attributes when assigning a new string to the Text property. You should instead use the SelText, SelStart, and SelLength to make changes to substrings within a styled EditField. (Using SelTextFont, etc. may change the entire appearance of the text — always true if the field is not styled — but does not change the "default" settings used when the entire text is replaced with the Text property.)
TextColor	Color	The color of the text (black by default).
TextFont	String	Name of the font used to display the text. Assigning another font to TextFont changes the font of all the text in the EditField. It also resets the default which will be used if the Text property is subsequently used to reset the contents of the text in the EditField.
TextSize	Integer	Size of the font used to display the text.
TextStyleData	String	The style data for Styled EditField. This data can be stored in a 'styl' resource and can be set with the SetTextAndStyle method.
Underline	Boolean	Applies the underline style to the text.
UseFocusRing	Boolean	If True, the object indicates that it has the focus with a ring around its border; if False, the appearance of the object does not change when it has the focus.

Events

Name	Parameters	Description
GotFocus		The cursor has moved into the EditField.
KeyDown	Key as String	The user has pressed the Key passed while the EditField has the focus. Returns a Boolean. Returning True means that no further processing is to be done with *Key*.
LostFocus		The cursor has left the EditField.

Name	Parameters	Description
MouseDown	x as Integer, y as Integer	The mouse button was pressed inside the control's region at the location passed in to x,y. Return True if you are going to handle the MouseDown. Returning True prevents the EditField from handling the mouse click.
MouseUp	x as Integer, y as Integer	The mouse button was released inside the control's region at the location passed in to x,y. This event will not occur unless you return True in the MouseDown event.
SelChange		The range of characters highlighted has changed.
TextChange		The text property of the EditField has been changed.

Methods

Name	Parameters	Description
BindValue	value as StringProvider	Used in implementing custom object bindings.
CharPosAtLineNum	LineNumber as Integer	Returns as Integer the character position in an Editfield of the first character on the *LineNumber* line. Characters are numbered consecutively from the start until the end of the EditField. The first character is numbered 1. The first line is numbered zero.
CharPosAtXY	X as Integer Y as Integer	Returns as Integer the character position for pixel coordinates *X*, *Y* relative to the EditField. Characters are numbered consecutively from the start until the end of the EditField. The first character is numbered 1.
Copy		Copies the selected text in the EditField to the Clipboard, including the styled text information.
LineNumAtCharPos	CharPosition as Integer	Returns (as Integer) the line number in which the *CharPosition* character appears. Characters are numbered consecutively with the first character numbered 1. The first line is numbered zero. Used with ScrollPosition, this lets you scroll the EditField to a particular place in the text.
Paste		Pastes the styled or unstyled text on the Clipboard into the EditField at the insertion point, adding the text to the existing text.
SetFocus		Gives the EditField the focus, sending all keydown events to the EditField and moving the cursor there.

Name	Parameters	Description
SetTextAndStyle	Text as String, TextStyleData as String	Places the text into the Text property of the EditField and applies the styles to it based on the data in TextStyleData. *TextStyleData* is compatible with the style resource.
StyledTextPrinter	g as Graphics, Width as Integer	Used to create StyledTextPrinter object for printing the EditField's text property as styled text. The EditField's MultiLine property must be True to support styled text printing. Returns a StyledTextPrinter Object. Width (pixels) is width of printable area.
ToggleSelectionBold		Reverses the bold style of the highlighted text.
ToggleSelectionCondense		Reverses the condensed style of the highlighted text. Not supported on Windows.
ToggleSelectionExtend		Reverses the extended style of the highlighted text. Not supported on Windows.
ToggleSelectionItalic		Reverses the italic style of the highlighted text.
ToggleSelectionOutline		Reverses the outline style of the highlighted text.
ToggleSelectionShadow		Reverses the shadow style of the highlighted text.
ToggleSelectionUnderline		Reverses the underline style of the highlighted text.

Notes

Working With Styled Text in EditFields

In order to display styled text in an **EditField**, the **EditField**'s Styled property must be checked. The SelBold, SelTextColor, SelCondense, SelExtend, SelItalic, SelOutline, SelShadow and SelUnderline properties can be use to both set the style of the selected text and get the style of the selected text (The Outline, Shadow, Condense, and Extend styles are not available on Windows). These properties are set automatically by REALbasic at runtime when text is selected in an **EditField**. To set the style, simply assign True to the property. For example, if you want to add the bold style to the selected text in EditField1, use this syntax:

```
EditField1.SelBold=True
```

You can determine if the selected text is in a particular style using the same property. The following code plays a beep sound if the selected text is bold:

```
If EditField1.SelBold Then
  Beep
End If
```

The SelBold, SelCondense, SelExtend, SelItalic, SelOutline, SelShadow and SelUnderline will be True if all of the selected characters are in the style being checked. Otherwise, these properties will be False.

Getting and setting the selected text font and font size work the same way using the SelTextFont and SelTextSize properties. To set the font of the selected text, set the SelTextFont property to the name of the font. If the selected text uses more than one font, SelTextFont will be an empty string. If the selected text has more than one font size, the SelTextSize property will be zero.

If the selected text has more than one style, font, or size, you can select individual characters using the SelStart and SelLength properties to determine which styles, fonts, and sizes are in use. Please note that if an EditField contains more than one style, font, or fontsize and you reassign the value of the Text property, you will lose the styled formatting of the text substrings. All the text will then be styled uniformly. To update the text in a styled EditField, it is safest to use the SelStart, SelLength, and SelText properties to update text selections.

You can also get and set the color of the selected text using the SelTextColor property. for example, the following code checks to see if the selected text is white and if it is, sets it to black:

```
Dim White, Black as Color
White=RGB(255,255,255)
Black=RGB(0,0,0)
If EditField1.SelTextColor=White Then
  EditField1.SelTextColor=Black
End If
```

To print styled text in an **EditField**, use the StyledTextPrinter method to create a StyledTextPrinter object and then use the StyledTextPrinter's DrawBlock method. See the description of StyledTextPrinter for an example. There is also an example of styled text printing in the Tutorial lesson on styled text.

Styled Text on Windows

The following styles are not supported on the Windows platform: Outline, Shadow, Condensed, Extended.

Single-line EditFields

Single-line EditFields on Mac OS 9 (or earlier) have a limit of 32K of text. They also support extended ASCII characters and the Symbol font.

Adding Text to an EditField

When appending text to an **EditField**, you may notice some flicker as REALbasic redraws the **EditField** to show the new text. This will happen if you appended the Text property of the **EditField** like this:

```
EditField1.Text=EditField1.Text+"my new text"
```

This occurs because the entire contents of the EditField has to be redrawn. To avoid this flicker, position the cursor at the end of the **EditField** and append the text using the SelText property like this:

```
EditField1.SelStart=Len(EditField1.Text)
EditField1.SelText="my new text"
```

Text Encoding in Carbon

Under Carbon/Mac OS X, EditFields store all text internally in Unicode, which is able to represent a mixture of fonts from different writing systems. When you extract the text via the Text or SelText properties, this Unicode text is converted according to the encoding associated with the TextFont of the field. For example, if TextFont is "Osaka," the text will be converted to MacJapanese. Two special values for TextFont allow you to access the Unicode content directly. If TextFont is "System.UTF8" then the text is returned in UTF-8 encoding. If it is "System.UTF16" then the text is return as UTF-16. Both of these can represent the full Unicode character set.

See Also

Font, FontCount functions; StyledTextPrinter class.

EmailAttachment Class

Used to hold attachments to an email.

Super Class

Object

Properties

Name	Type	Description
ContentEncoding	String	Content encoding of the attachment.
Data	String	When receiving an attachment, the contents of the attachment.
MacCreator	String	Four-character Macintosh Creator code of the attached file.
MacType	String	Four-character Macintosh Type code.
MIMEType	String	MIME type of the attachment.
Name	String	Name of the attached file.

Methods

Name	Parameters	Description
LoadFromFile	File as FolderItem	Attaches the passed *File*.
SaveToFile	File as FolderItem	Saves the attachment to *File*. Returns a Boolean, True if successful.

Example

The following example takes a path to a file and adds it as an email attachment. The path has been entered into the EditField named fileFld.

```
Dim file as EmailAttachment
Dim mail as EmailMessage
If fileFld.text <> "" then
 file = New EmailAttachment
 file.loadFromFile GetFolderItem(fileFld.text)
 mail.Attachments.Append file
end if
```

See Also

EmailMessage, EmailHeaders, POP3Socket, POP3SecureSocket, SMTPSocket, SMTPSecureSocket, TCPSocket, SocketCore classes.

EmailHeaders Class

Used to hold the headers for an email.

Super Class

Object

Methods

Name	Parameters	Description
AppendHeader	Name as String, Value as String	Appends a new header.
Count		Returns the number of headers as an Integer.
Delete	Name as String OR Name as String, Index as Integer	Deletes the specified email header.
Name	Index as Integer	Returns as a String the name of the specified header.
SetHeader	Name as String, Value as String	Sets the name and value of a header.
Source		Returns a String containing the raw source text of the headers.
Value	Name as String OR Name as String, Index as Integer OR Index as Integer	Returns a String containing the value of the specified header.

Name	Parameters	Description
ValueCount	Name as String	Returns an Integer. Gets the number of header values that have the name specified by *Name*. This is included because it is possible to have more than one header with the same name.

Example

The following sets the header of an outgoing email.

```
Dim mail as EmailMessage
mail=New EmailMessage
mail.headers.AppendHeader "X-Mailer","REALbasic SMTPSocket Demo"
```

See Also

EmailMessage, EmailAttachment, POP3Socket,POP3SecureSocket, SMTPSocket, SMTPSecureSocket, TCPSocket, SSLSocket, SocketCore classes.

EmailMessage Class

Used to hold the contents of an email message and its enclosures, if any.

Super Class

Object

Properties

Name	Type	Description
Attachments	EmailAttachment	The array of EmailAttachments that are attached to this email message.
BCCAddress		Returns an array of Strings. Gets the BCC addresses for this email
BodyEnriched	String	The body text of the email message in enriched format. Some clients such as Mail use this format when sending styled emails.
BodyHTML	String	The body text of the email as HTML.
BodyPlainText	String	The body text of the email as plain text. Clients that do not support HTML or enriched format will display the message as plain text using the contents of this property.
CCAddress		Returns an array of Strings. Gets the CC addresses for this email.
FromAddress		Returns a String containing the address the email is coming from.
FromAddress	String	Sets the From address for this email message.
Headers	EmailHeaders	The headers for this email message.

Name	Type	Description
Source		Returns a String, containing the raw source of the email message.
Subject		Returns a String. Gets the subject of the email.
Subject	String	Sets the subject of this email.
ToAddress		Returns an array of Strings. Gets the addresses for this email.

Methods

Name	Parameters	Description
AddBCCRecipient	Recipient as String	Adds a BCC recipient to the email.
AddCCRecipient	Recipient as String	Adds a CC recipient to the email.
AddRecipient	Recipient as String	Adds a recipient to the email.

Example

The following example sets the values of the From Address, Subject, Body, and Headers from the values of the Text properties of EditFields in a window. The EditFields named fromAddressfld, subjectFld, bodyFld, and htmlFld contain this information:

```
Dim mail as EmailMessage
mail=New EmailMessage
mail.fromAddress=fromAddressFld.text
mail.subject=subjectFld.text
mail.bodyPlainText = bodyFld.text
mail.bodyHTML = htmlFld.text
```

See also the example for the SMTPSocket class.

See Also

EmailHeaders, EmailAttachment, HTTPSocket, HTTPSecureSocket, POP3Socket, POP3SecureSocket, SMTPSocket, SMTPSecureSocket, TCPSocket, SSLSocket, SocketCore classes.

EnableMenuItems Method

Forces an update of the menu bar and executes the EnableMenuItems event handler.

Syntax

EnableMenuItems

Notes

In most cases, REALbasic's standard menu bar updating will be sufficient. There are cases, however, when it's not. For example, if the SelChange event handler of an EditField control causes all of the menu items in a particular menu to be disabled, the

menu itself should be disabled. Unfortunately, it won't be because the user has not yet clicked in the menu bar. Under circumstances like this, you can call the **EnableMenuItems** method to force REALbasic to check the conditions of the menus and redraw them if necessary.

EncodeBase64 Function

Converts a string to Base64 format.

Syntax

***result*=EncodeBase64(*str*)**

Part	Type	Description
result	String	The Base64 representation of *str*.
str	String	The String expression to be converted to Base64 format.

Notes

Base64 is used for email file attachments. SMTP was designed around ASCII, so it expects that characters below ASCII 32 to be control codes and doesn't expect anything above ASCII 127. Base64 lets you encode arbitrary binary data into a 64-character alphabet composed of the printable ASCII characters. Three bytes of input become four characters of output. Base64 is also used in authentication schemes as a way to send an encryption key or hash value through a link that expects text.

The DecodeBase64 function performs the reverse operation, taking a Base64 expression and converting it to the original data.

Example

The following code displays the ASCII and Base64 codes corresponding to the numbers 0 to 255 in a ListBox.

```
Dim s as String
Dim i as Integer
For i=1 to 256
 ListBox1.addrow Str(i-1)
 ListBox1.cell(i-1,1)=Chr(i-1)
 ListBox1.cell(i-1,2)=EncodeBase64(Str(i-1))
Next
```

See Also

DecodeBase64 function.

EncodeQuotedPrintable Function

Converts a String into quoted-printable format.

Syntax

***result*=EncodeQuotedPrintable(*str*)**

Part	Type	Description
result	String	The result of processing *str*.
str	String	The string expression to be converted.

Notes

EncodeQuotedPrintable converts unprintable characters (i.e., control characters, returns and tabs) into hexadecimal. For example, if you pass it a string that contains return characters at the end of each line, they will be replaced by "0D" in the result.

The DecodeQuotedPrintable function performs the function in reverse.

See Also

DecodeQuotedPrintable function.

Encoding Function

Returns the text encoding of the passed String.

Syntax

***result*=Encoding(*str*)**

Part	Type	Description
result	TextEncoding	The text encoding of *str*.
str	String	The string whose TextEncoding will be returned.

Notes

If the string's encoding is unknown, **Encoding** returns Nil.

Example

The following returns the encoding of the contents of an EditField.

```
Dim t as TextEncoding
t=Encoding(EditField1.text)
```

See Also

TextEncoding, TextConverter classes, GetTextEncoding function; Encodings object.

Encodings Object

Returns the specified TextEncoding.

Syntax

result*=Encodings.*EncodingsName

Part	Type	Description
result	TextEncoding	The text encoding specified by *EncodingsName*.
EncodingsName OR	String	The name of a TextEncoding. See Appendix A of the printed or pdf version of the *Language Reference*, or preferably, use the Code Editor's AutoComplete feature to show all possible values of *EndodingsName*.
GetFromCode		Parameter is *Code* as Integer and returns a TextEncoding object. Values for Code are those used by TextEncoding. This provides a convenient way to save a TextEncoding to a file, send it over a socket or serial port, and so forth.

Notes

The **Encodings** object makes it easy to obtain a specified TextEncoding. Any text encoding can be obtained via the **Encodings** object. Some of the most useful are UTF8, UTF16, UCS4, ASCII, MacRoman, MacJapanese, and WindowsLatin1. Use the Autocomplete feature of the Code Editor to view the complete list.

Example

The following statement uses the **Encodings** object to obtain the UTF8 text encoding.

```
Editfield2.text=DefineEncoding(EditField1.text,Encodings.UTF8)
```

See Also

TextEncoding class, DefineEncoding, Encoding functions.

End Quote Missing Error

An end quote was missing from a literal string.

Example

An end quote is missing from the following String passed to the MsgBox command:

```
MsgBox "Hello world
```

EndOfLine Object

Returns an end-of-line terminator.

Syntax

***result*=EndOfLine**

Part	Type	Description
result	String	The end of line String for the platform being compiled.

or

***result*=EndOfLine.EndOfLineType**

Part	Type	Description
result	String	The end of line String specified by *EndOfLineType*.
EndofLIneType		The end of line String being requested. The choices are: Macintosh Windows Unix

Notes

Macintosh uses the ASCII 13 (Return) to indicate the end of a line of text; Windows uses ASCII 13 (Return) + ASCII 10 (Line Feed) to indicate the end of a line.

Use the ReplaceLineEndings function to convert the line endings of text from one line ending to another.

Examples

The following example uses the **EndOfLine** function to break text to be displayed in a MsgBox into two paragraphs.

```
MsgBox "Hello world"+EndOfLine+"Such as it is"
```

The following example defines a local variable as the correct line ending for each platform the application will run on and uses it in a text string.

```
Dim cr as String
#if TargetMacOS
  cr=EndOfLine.Macintosh
#else
  cr=EndOfLine.Windows
#endif
Editfield1.Text ="Hello world"+cr+"Such as it is."+cr
```

See Also

ReplaceLineEndings function.

Exception Block

Used to handle RuntimeException errors.

Syntax

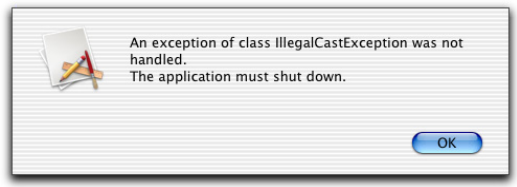
Exception [*ErrorParameter*] [As *ErrorType*]

Part	Description
ErrorParameter	Optional, used to determine the type of runtime exception.
ErrorType	Optional, if used, it must be used with *ErrorParameter*. Used to 'catch' a particular type of runtime error. If used, the Exception block will handle only that type of runtime error.

Notes

One or more Exception blocks can be inserted after the last 'regular' line of code to catch and handle runtime exceptions that may occur anywhere within the method.

If a runtime exception occurs in a built application and is not handled, REALbasic will display a generic runtime error message box and quit the application. An example message box is shown below:



Exception blocks provide a means of handling the error more gracefully.

Exception blocks always appear at the end of a method (not where you think the error might occur) because every line after the **Exception** line is considered part of the exception block. In the Code Editor, the **Exception** line has the same level of indentation as the Sub or Function line.

You can use **Exception** alone if you wish to handle any type of exception in the Exception block, as shown below:

```
Sub...
 .
 .
Exception
 MsgBox "Something really bad happened, but I don't know what."
End Sub
```

The example shown above is sufficient to prevent the application from quitting, but the message is not very informative because you don't have a clue what type of exception occurred.

The way to determine which type of runtime error occurred is to use *ErrorParameter* and, in some way, test its type. *ErrorParameter* can be any of the following error types.

ErrorType	Description
IllegalCastException	You attempted to recast an object and sent it a message its real class can't accept.
KeyNotFoundException	You tried to access an item in a Dictionary using a key that is not in the Dictionary.
NilObjectException	An attempt was made to access an object that does not exist.
OutOfBoundsException	An attempt was made to read from or write to a value, character, or element outside the bounds of the object or data type, i.e., you tried to access an array element that doesn't exist.
OutOfMemoryException	Raised in certain cases when you run out of memory, for example when there isn't enough memory to complete an operation.
RegExException	You used an incorrect syntax in a regular expression.
RegExSearchPatternException	You specified an incorrect search pattern in a regular expression.
ShellNotRunningException	You tried to access an interactive shell when it wasn't running. or tried to change the context of the script while it was running.
StackOverflowException	When one routine (method/event handler/menu handler) calls another, memory is used to keep track of the place in each routine where it was called along with the values of its local variables. The purpose of this is to return (when the routine being called finishes) to the previous routine with all local variables as they were before. The memory set aside for tracking this is called the Stack (because you are "stacking" one routine on top of another). If your application runs out of stack space, a StackOverflowException will occur.
TypeMismatchException	You tried to assign to an object the wrong data type. Occurs only when using an object that REALbasic cannot type at compile-time.
UnsupportedFormatException	You used an incorrect syntax in an expression, for example for column widths in a ListBox, or tried to use an unsupported format in saving and opening picture.s

If you install the Office Automation plug-ins, you also add WordException, ExcelException, and PowerPointException types. Other optional plug-ins may also add their own exception types.

One way to test *ErrorParameter* is with an If statement in the Exception block:

```
Sub…
 .
 .
Exception err
 If err IsA TypeMismatchException then
  MsgBox "Tried to retype an object!"
 elseif err IsA NilObjectException then
  MsgBox "Tried to access a Nil object!"
 .
 .
 End if
End Sub
```

Instead of using multiple If statements, you can use multiple Exception blocks, each of which handles a different runtime exception type:

```
Sub…
 .
 .
Exception err as TypeMismatchException
 MsgBox "Tried to retype an object!"
Exception err as NilObjectException
 MsgBox "Tried to access a Nil object!"
End Sub
```

Examples

See the examples for the NilObjectException, IllegalCastException, TypeMismatchException, OutOfBoundsException, and StackOverFlowException errors.

See Also

Function statement, Raise statement, NilObjectException, IllegalCastException, KeyNotFoundException, TypeMismatchException, OutOfBoundsException, OutOfMemoryException, RegExException, RegExSearchPatternException, StackOverFlowException, ShellNotRunningException, UnsupportedFormatException errors; RuntimeException class; Nil function.

Exception Objects Must be of Type RuntimeException Error

You declared an exception variable as something other than a RuntimeException or a subclass of RuntimeException.

Example

The following declaration produces this error:

```
Exception error as Double
```

Exit Statement

The Exit statement causes control to continue at the statement following the end of a loop without the loop conditions being satisfied.

Syntax

Exit

Example

This example increments a counter variable. If the user presses Command-Period, the UserCancelled function will return True and the loop will exit. Otherwise, the loop will continue until the Stop property is True:

```
Dim counter as Integer
Do
 counter=counter+1
 If UserCancelled then
  Exit
 End if
Loop Until Stop
```

See Also

Do...Loop, For...Next, While...Wend, GOTO statements.

Exp Function

Returns "e" to the power of the value specified.

Syntax

***result*=Exp (*value*)**

Part	Type	Description
result	Double	The exponential of *value*.
value	Double	The value you want the exponential of.

Notes

The **Exp** function returns "e" to the power of the value passed to it.

Examples

This example uses the **Exp** function to return the exponential of a number.

```
Dim d as Double
d=Exp(10) //returns 22026.4657948
```

A space rather than a comma is used to separate variable names in a Dim statement:

```
Dim a b as Integer //should be 'a,b'
```

See Also

Dim statement.

ExportPicture Function

Exports an object of type Picture.

Syntax

***result*=ExportPicture(*pic*)**

Part	Type	Description
result	Boolean	True if the picture was saved successfully; False if the picture was not saved.
pic	Picture	A picture.

Notes

When **ExportPicture** is called, it displays a save-file dialog box so the user can save the picture in any directory. The user can also use the dialog box to choose a picture format. The available choices are: BMP, PICT, PhotoShop, JPEG, PNG, SGI, TGA, TIFF, and QuickTime image. When the user chooses a format, REALbasic adds the appropriate suffix to the filename.

The PICT and BMP formats are supported on Macintosh and Windows, respectively. For all other formats, QuickTime is required.

ExportPicture returns True if the picture was saved; if the user clicks cancel, ExportPicture returns False.

Example

The following code allows the user to save a picture.

```
Dim p as Picture
Dim r as Boolean
Dim f as FolderItem
f=GetFolderItem("IconView")
p=f.OpenAsPicture
r=ExportPicture(p)
If r then
  MsgBox "Your picture was saved."
else
  MsgBox "Your picture was not saved."
end
```

See Also

Picture, FolderItem classes.

Extends Keyword

Used in a method declaration to indicate that the method is to be called using the object's syntax, i.e., as a method belonging to the object.

Syntax

Extends *parameter* As *Object*

Part	Type	Description
parameter		The name of the first parameter.
Object	Any valid Object type	The Object type from which the new method is to be called.

Notes

The **Extends** keyword enables you to call a user-defined method as a method belonging to an object. You use the Extends keyword only for the first parameter in the method declaration. **Extends** indicates that this parameter is to be used on the left side of a dot operator (".").

Methods declared in this way are sometimes called "class extension methods." You can use **Extends** only for methods in a Module.

Example

To call a new method, myNewFolderItemMethod as a method of the FolderItem class, use a declaration such as:

```
Sub myNewFolderItemMethod(Extends f as FolderItem)
```

You can then call it in code such as:

```
Dim f as FolderItem
f.myNewFolderItemMethod
```

See Also Extends can only be used on the first parameter Error.

Extends can only be used on the first parameter Error

You used the Extends keyword more than once and/or on a parameter other than the first parameter.

Example

```
Sub myNewMethod(x as Integer, Extends f as FolderItem)
```

Should be rewritten as:

```
Sub myNewMethod(Extends f as FolderItem, x as Integer, )
```

See Also Extends keyword.

ExtensionsFolder Function

Used to access items in the Extensions folder in the System folder. On Windows, it accesses the WINDOWS\SYSTEM directory.

Syntax

***result*=ExtensionsFolder**

Part	Type	Description
result	FolderItem	A FolderItem that represents the Extensions folder.

Notes

Use the **ExtensionsFolder** function to access the Extensions folder.

The **ExtensionsFolder** function provides a way to access the Extensions folder that will work under different language systems, and will continue to work even if Apple reorganizes the System folder. If your application will run under Windows, you can check the TargetWin32 global constant before calling **ExtensionsFolder**.

Examples

See the AppleMenuFolder function for an example.

See Also AppleMenuFolder, ApplicationSupportFolder, ControlPanelsFolder, FontsFolder, PreferencesFolder, StartupItemsFolder, SystemFolder, TemporaryFolder, TrashFolder functions.

External Functions Cannot Use Objects as Parameters Error

This error occurs if you Declare an external function and try to specify that one of its parameters is an object.

When passing a MemoryBlock to a Declare, use Ptr instead. When passing a window to a Declare, use WindowPtr instead.

See Also Declare statement, MemoryBlock class.

External Functions Cannot Use String Data Types as Parameters Error

Occurs if one of your Declare statement's parameters is an ordinary REALbasic string. Use CString or PString instead.

See Also Declare statement.

False Function

Used to set Boolean values.

Syntax ***expression*=False**

Expression is any valid Boolean expression.

See Also True function, And, Or, and Not operators.

FigureShape Class

Used to draw vector graphic shapes composed of lines and curves. It is similar to a polygon except that a 'side' may be curved. A figure consists of a set of CurveShapes, with the end point of one curve automatically joined to the starting point of the next.

Super Class

Object2D

Properties

Name	Type	Description
Count	Integer	The number of CurveShapes.
Item	CurveShape	Parameter is Index as Integer. Gets or sets the specified curve.

Methods

Name	Parameters	Description
AddCubic	X as Integer Y as Integer X2 as Integer Y2 as Integer CX as Integer CY as Integer CX2 as Integer CY2 as Integer	Adds a cubic curve to the figure. The starting and ending points are x,y and x2, y2, respectively. The parameters CX and CY is the 'power' of the quadratic expression. The parameters CX2 and C) Y2 are the third-degree power terms.
AddLine	X as Integer Y as Integer X2 as Integer Y2 as Integer	Adds a straight line to the figure. X and Y are the starting coordinates; X2 and Y2 are the end coordinates.
AddQuad	X as Integer Y as Integer X2 as Integer Y2 as Integer CX as Integer CY as Integer	Adds a quadratic curve to the figure. The starting and ending points are x,y and x2, y2, respectively. The parameters CX and CY is the 'power' of the quadratic expression.
Append	Curve as CurveShape	Adds the passed CurveShape to the figure.
Insert	Index as Integer Curve as CurveShape	Inserts the passed CurveShape at the position indicated by *Index*.
Remove	Index as Integer OR Curve as CurveShape	Removes a curve, specified either by its index or by reference.

Notes

A figure is formed by drawing each curve it contains in order, joining the endpoint of one to the starting point of the next. The figure is degenerate if doing so does not

enclose any area. This will be the case for a figure containing only one line, for example. The appearance of a degenerate figure is undefined.

AddLine, AddCubic, and AddQuad are convenience methods that make it easier to add curves to the figure. You could instead create your own CurveShapes, and add them with the Append or Insert methods.

Example

The following draws a square from two lines:

```
Dim c,d as CurveShape
Dim fx as FigureShape
fx=New FigureShape
c=New CurveShape
d=New CurveShape
c.controlx(0)=120
d.controlx(0)=120
c.controly(0)=40
d.controly(0)=40
c.order=0
d.order=0
c.x=10
c.y=10
c.x2=100
c.y2=10
d.x=100
d.y=100
d.x2=10
d.y2=100
fx.append c
fx.append d
fx.Border=100   //opaque border
fx.BorderColor=&c000000    //black
fx.FillColor=&cFF0000  //red
Graphics.DrawObject fx,50,50
```

See Also

ArcShape, CurveShape, FolderItem, Group2D, Graphics, Object2D, OvalShape, Picture, PixmapShape, RectShape, RoundRectShape, StringShape classes.

FillColor Function

The currently selected Appearance Manager background color.

Syntax

***result*=FillColor**

Part	Type	Description
result	Color	The color used for drawing the background of controls and windows.

Notes

This value is useful when you are using Canvas controls to create custom controls that you want to be Appearance Manager savvy. When drawing objects, use this color to fill the background.

This value can be changed by the user on the fly using the Kaleidoscope Control Panel, so you should access this value during your Paint event handler rather than storing the value.

See Also

DarkBevelColor, DarkTingeColor, FrameColor, LightBevelColor, LightTingeColor, HighlightColor, TextColor functions, Color data type.

Floor Function

Returns the value specified rounded down to the nearest Integer.

Syntax

***result*=Floor (*value*)**

Part	Type	Description
result	Double	The floor of *value*.
value	Double	The value you want the floor of.

Notes

The **Floor** function returns the value passed to it rounded down to the nearest Integer.

Example

This example uses the **Floor** function to return floor of a number.

```
Dim d as Double
d=Floor(1.234) //returns 1
```

See Also

Ceil function, Round function.

FolderItem Class

FolderItem class objects represent files, applications, or folders. They are created by calling a method such as GetFolderItem or via the Parent or Item properties of other **FolderItem** objects.

Super Class Object

Properties

Name	Type	Description
AbsolutePath	String	The full path to the **FolderItem**. On Macintosh, the pathname ends with a trailing colon; on Windows, it ends with a trailing backslash. When accessing a drive on Windows that you do not have permissions for or does not exist, AbsolutePath has no trailing slash. For example, "a:" is a floppy drive with no disk, "a:\" has a disk.
Alias	Boolean	Returns True if the item is an alias (Macintosh only). On Windows, Alias always returns False.
Count	Integer	The number of items in the **FolderItem** (if it's a directory/folder).
CreationDate	Date	The creation date of the **FolderItem**.
DesktopFolder	**FolderItem**	Returns a **FolderItem** that references the Desktop Folder belonging to the parent volume.
Directory	Boolean	The **FolderItem** is a directory (a folder).
DisplayName	String	The name of the **FolderItem** as it should be seen by the user. Usually the same as the Name property. Under Mac OS X 10.1, it could be the name without the extension, depending on whether the user has file extensions hidden for that **FolderItem**. Under Mac OS X, use DisplayName rather than Name when displaying the name of the item to the user.
ExtensionVisible	Boolean	Allows you to tell whether a given file has its file extension hidden or visible, or toggle that status. Currently applies only under Mac OS X 10.1 or above. Note that when you change this setting, the Finder or subsequent Navigation dialogs may not reflect the change right away. When you create a new file under Mac OS X, REALbasic sets the "hide extension" flag according to the current value of this property.
Exists	Boolean	Indicates whether or not the AbsolutePath property points to a file or directory that exists.
IsReadable	Boolean	True if it is possible to read from the **FolderItem**.

Name	Type	Description
IsWriteable	Boolean	True if it is possible to write to the **FolderItem**.
LastErrorCode	Integer	Contains the error code for the last supported operation on the **FolderItem**. The supported operations are: -CopyFileTo -MoveFileTo -Delete -Methods that create a file or folder The following error codes may be returned: 100: Destination does not exist. You will get this error only on CopyFileTo and MoveFileTo. 101: File not found. 102: Access denied 103: Out of memory 104: File in use 105: Invalid name LastErrorCode may also return operating system-specific error codes if the error does not map to one of the above errors.
Length	Integer	The size of the file's data fork. For folders, the size will be zero.
Locked	Boolean	The **FolderItem** is locked or resides on a locked volume.
MacCreator	String	For documents, the 4 character code that identifies the application that would be launched and would open the FolderItem if it were opened from the desktop. Assign the 4-character string to this property. If the FolderItem refers to a directory on Mac OS, MacCreator returns "MACS". MacCreator is Macintosh-specific.
MacDirID	Integer	Mac directory ID. Macintosh only.
MacType	String	The 4 character code that identifies the type of file (Macintosh only). For example, "APPL" is the type of all applications. "TEXT" is the type of most text documents. Assign the 4-character string to this property. If the **FolderItem** refers to a directory on Mac OS, MacType returns "fold"
MacVRefNum	Integer	Mac volume reference number (Macintosh only).
ModificationDate	Date	The modification date of the **FolderItem**.
Name	String	The name of the **FolderItem**. Changing this name will change the name of the file or folder. It will rename files if the user has permission to rename them, they are not in use, and it's not a folder or temporary file.
Parent	**FolderItem**	The **FolderItem** object for the parent of this item. Nil if this item is the root.
ResourceForkLength	Integer	The number of bytes in the file that make up the resource fork. Access to the resourcefork is supported only on Macintosh.

Name	Type	Description
SharedTrashFolder	FolderItem	Returns a **FolderItem** that references the shared trash folder belonging to the parent volume (Macintosh). Returns Nil on Windows.
Temporary Folder	FolderItem	Returns a **FolderItem** that references the temporary folder belonging to the parent volume.
TrashFolder	FolderItem	Returns a **FolderItem** that references the shared trash folder belonging to the parent volume (Macintosh). Returns Nil on Windows.
Type	String	If the **FolderItem** is not a directory, this property contains the file type of the FolderItem. If the FolderItem is a directory (Mac OS), it will return a matching file type if you have created one.
VirtualVolume	VirtualVolume	If the **FolderItem** is in a VirtualVolume, it returns the VirtualVolume that stores the file. If it is a real file, it returns Nil.
Visible	Boolean	True if the **FolderItem** is visible and False if it is not.

Methods

Name	Parameters	Description
AppendToTextFile		Returns a TextOutputStream. Appends the text to the text file. If the file does not exist, it is created.
Child	Name as String	If this is a directory, Child returns a **FolderItem** object for the item in this directory with the name passed. Aliases are resolved automatically (Macintosh). Child does not resolve aliases on Windows.
CopyFileTo	Item as **FolderItem**	If *Item* is a folder, then the **FolderItem** is copied into *Item*. If *Item* is a file and the file already exists, the copy is aborted. You need to delete the existing file first. If there is an error, the LastErrorCode property contains an error code.
CreateAsFolder		Creates a folder at the location specified by the **FolderItem** properties.
CreateBinaryFile	FileType As String	Creates a binary file of type FileType. FileType must be the name of a file type in the File Types dialog box. Returns a BinaryStream.
CreateMovie		Creates a new EditableMovie. Returns an EditableMovie object.
CreateResourceFork	FileType As String	Macintosh only. Creates a new resource fork. If file exists, then existing resource fork is emptied, otherwise new file is created with *FileType* passed. *FileType* is the name of a file type that was created in the File Types dialog box, not a Macintosh 4-character file type.

Name	Parameters	Description
CreateTextFile		Creates a text file at the location specified by the **FolderItem**. Returns a TextOutputStream.
CreateVirtualVolume		Creates and formats a new VirtualVolume at the location specified by the **FolderItem**. Returns a VirtualVolume.
Delete		Deletes the file or directory specified by the **FolderItem**. This method permanently removes the file or directory from the volume it's stored on. If you are deleting a directory, it must be empty. If there is an error, the LastErrorCode property contains an error number.
Item	Index as Integer	If this **FolderItem** is a directory, *Index* is an element in a one-based array of FolderItems in this directory. If the FolderItem is an alias, Item automatically resolves the alias and returns a FolderItem for the original file, folder, or application.
GetSaveInfo	RelativeTo as **FolderItem** Mode as Integer	Allow saving **FolderItem** references without relying on the absolute path. Returns a String. The returned string indicates a relative path with is relative to the folder passed in *RelativeTo*. Values for Mode are: 0: The **FolderItem** should be open in any way we can possibly create it. This is the Default value. 1: The **FolderItem** should only be opened using relative path information. 2: The **FolderItem** should only be opened using absolute path information. Note: The returned string is not intended to be human-readable and any modifications may render it useless.
GetRelative	SaveInfo as String	Returns a **FolderItem**. Returns a **FolderItem** based on the string passed to it. If the string indicates a relative path, the current **FolderItem** is considered its reference point. If the string passed to it is absolute, then the current **FolderItem** is ignored when resolving the path. GetRelative returns Nil only if there is not sufficient information in *SaveInfo* to construct a **FolderItem** (e.g., using a relative path that causes the parsing to descend below root level).
Launch	[Activate as Boolean]	If the **FolderItem** is an application, the application is launched. If the FolderItem is a document, the document is opened (and the application is launched if necessary). The optional parameter *Activate* specifies whether the application should be launched in the foreground or background. The default value is True. If you specify False, REALbasic will attempt to launch the application in the background, but this may not work with certain applications.

Name	Parameters	Description
MoveFileTo	Destination as **FolderItem**	Moves **FolderItem** to path specified by *Destination*. It does a move rather than a copy even when the source file is on another volume. If there is an error, the LastErrorCode property contains an error code.
OpenAsBinaryFile	Write as Boolean	Opens the **FolderItem** passed to be read as a binary file. If Write is True, the file is opened in read/write mode, otherwise it's opened in read only mode. Returns a BinaryStream.
OpenAsMovie		Opens the **FolderItem** to be read as a QuickTime movie and returns its movie as a Movie object. Nil is returned if the movie can't be read.
OpenAsPicture		Opens the **FolderItem** to be read as a PICT and returns its image as a picture object. Nil is returned if the image can't be read. If QuickTime is installed, OpenAsPicture will open any kind of graphics file QuickTime will open (JPEG, GIF, etc.)
OpenAsSound		Opens the **FolderItem** to be read as a Macintosh System 7 sound file and returns its contents as a sound object. On Windows, OpenAsSound recognizes MIDI, MP3, and MPEG files as valid sound formats. Note: to play MP3 files on Windows 95, check to see that you have the latest Windows Media Player installed. Nil is returned if the sound can't be read or isn't a sound file at all.
OpenAsTextFile		Opens the **FolderItem** to be read as a text file. Returns a TextInputStream.
OpenAsVectorPicture		Opens the **FolderItem** to be read as a vector Picture. It will do its best to convert a PICT file (on Macintosh) or an .emf file (Windows) to a Picture composed of Object2D objects. The original file may contain elements that do not have an equivalent Object2D object. REALbasic will do its best to map these objects, but there may be some loss of information, depending on the characteristics of the original file. PICTs support unrotated Rectangles, Lines, Ellipses, RoundRects, Polygons, Text, Pixmaps, and Arcs. .emf files support unrotated Rectangles, Lines, Ellipses, RoundRects, Polygons, Text, Pixmaps and Arcs. .emf files are displayed as actual size. For the most part this is huge; you probably will want to scale them down before viewing (pic1.objects.scale = scalingFactor). We find that a scale factor of 0.045 is a good value.
OpenAsVirtualVolume		Attempts to open the **FolderItem** as a VirtualVolume. It returns Nil if the **FolderItem** is not a VirtualVolume. it is opened as Read/Write unless it is locked at the operating system level. Returns a VirtualVolume.

Name	Parameters	Description
OpenEditableMovie		Opens the **FolderItem** as an EditableMovie. Nil is returned if the movie can't be read.
OpenResourceFork		Opens the resource fork of the **FolderItem**. Access to the resourcefork is supported only on Macintosh.
OpenResourceMovie	ResID as Integer	Opens the movie in ResID as a movie (Macintosh only). Used only if the movie is stored as a MooV resource. Use OpenAsMovie for QT 4.0 (and greater) files.
OpenStyledEditField	Field As EditField	Places the styled text from the **FolderItem** into the Field passed.
SaveAsJPEG	Picture as Picture	Saves the picture in JPEG format. Appends ".jpg" to the Windows filename. Requires QuickTime installed on the user's computer.
SaveAsPicture	Picture as Picture [Format as Integer]	Saves the picture in PICT format on Macintosh or BMP format on Windows and appends ".bmp" to filename. QuickTime is not required. The optional second parameter specifies the format: Specifying the proper format allows users to save vector data in a vector format rather than converting to bitmaps. See the Notes section for a description of the possible values of *Format*.
SaveStyledEditField	Field As EditField	Saves the styled text from the Field passed to the **FolderItem**. If you have defined a File Type of 'styledText' or 'text' then it will use those creator/type values for the saved file.
TrueChild	Pathname as String	If Pathname is a directory, TrueChild returns a **FolderItem** object for the directory with the name passed. TrueChild returns the actual **FolderItem**, even if it is an alias.
TrueItem	Index as Integer	If this **FolderItem** is a directory, *Index* is an element in a one-based array of FolderItems in this directory. TrueItem returns the actual **FolderItem**, even if it is an alias.

Notes

Specifying Pathnames

When building a cross-platform application in REALbasic that accesses files, you have to take into account the fact that Mac OS and Windows use different characters to separate volumes, directories, and filenames in a path. This is important to keep in mind when using the GetFolderItem method (which requires a pathname) or when using the AbsolutePath property of the **FolderItem** class (which contains the full path to the item).

One thing you can do to make using GetFolderItem a bit easier is to add a Separator property to your Application class and in its Open event assign the correct separator

based on the platform your application is compiled for. You can then use that Separator property when calling GetFolderItem.

A good cross-platform method for getting a **FolderItem** is to use the Volume function and the Child method of the **FolderItem** class. This avoids the separator issue completely. For example, to get a **FolderItem** for Microsoft Word in the Applications folder on the boot volume, you could use the following line of code:

```
Dim f as FolderItem
f= Volume(0).Child("Applications").Child("Microsoft Word")
```

Mac OS X is based on BSD Unix which uses "/" as the separator. However, because REALbasic and all applications made with REALbasic are Carbon applications on Mac OS X, they continue to use ":" as the separator. The one exception is when accessing files via the Shell class. In this case, you would use the same separator that you use on the command line.

Constructors

When you create a FolderItem with the New command, you can pass the full or relative path to the new FolderItem as an optional parameter. For example:

```
Dim f as FolderItem
f=New FolderItem("myDoc.txt")
```

specifies the name of the new FolderItem and it is located in the same folder as the REALbasic application (if you're running in the IDE) or the same folder as the built application.

You can also create a **FolderItem** without passing any parameters. It works the same as passing an empty text string.

Picture Formats

The SaveAsPicture method has an optional second parameter that enables you to specify the format in which the picture will be saved. Formats in the range of 0-99 are reserved for meta-formats. Meta-formats map to a various concrete formats based on the target, the data being saved, or other criteria (The mappings may change in future versions of REALbasic).

- The numbers 100-199 are reserved for formats that are supported on multiple platforms (currently none).
- 200-299 are reserved for formats that are supported on the Macintosh.
- 300-399 are reserved for formats supported on Windows.
- x00-x49 are vector formats, while x50-x99 are raster formats.

MetaFormats

Value	Compatibility	Description
0	MostCompatible	Most widely-used format for the platform Mac = PICT Win32 = BMP)
1	MostComplete	Format most likely to retain all vector info Mac = PICT Win32 = EMF)
2	Default	DefaultVector or DefaultRaster, depending on picture data Mac = PICT Win32 vector=EMF Win32 raster= BMP
3	DefaultVector	Platform's standard vector format Mac = PICT Win32 = EMF)
4	DefaultRaster	Platform's standard raster format Mac = Raster PICT Win32 = BMP)

Macintosh Formats

Value	Format	Description
100	PICT	Includes simple vector data.
250	RasterPICT	Flattens all vector data to pixels.

Win32 Formats

Value	Format	Description
300	WMF	Windows Metafile format (old vector format).
301	EMF	Extended Metafile format (newer vector format).
350	BMP	Windows bitmap format.

Cross Platform Formats

No cross-platform formats are currently implemented.

Unrecognized formats or formats not supported for the built target will result in an UnsupportedFormatException. The Message property of the exception will contain additional information in as to what went wrong.

Aliases

If a **FolderItem** is actually an alias to a **FolderItem**, the alias is automatically resolved when the **FolderItem** is accessed unless you use TrueChild and TrueItem (Aliases for Win32 do not get resolved with Item and Child). They return the item itself, even if it is an alias. Use the Alias property to determine whether the FolderItem is an alias.

For more information on File Types, see the chapter called "Working With Files" in the *User's Guide*.

Movies

The OpenResourceMovie method allows you to get a movie stored in a MooV resource inside a file or application. Prior to QuickTime 4.0, the actual movie in a QuickTime movie file was stored in a MooV resource. In QuickTime 4.0 (and above), the movie is stored in the data fork.

Windows Considerations

The AbsolutePath property returns a trailing colon on Macintosh and a trailing backslash on Windows.

Examples

This example puts the names of all the items on the Desktop that are stored on the boot volume into ListBox1.

```
Dim i,n  as Integer
Dim f as FolderItem
f=Volume(0).DesktopFolder
n=f.count
If n>0 then
 For i=1 to n
  ListBox1.addrow f.item(i).name
 Next
End if
```

This example uses MoveFileTo. The source file will be deleted and moved into the destination folder.

```
Dim f as FolderItem
Dim g as FolderItem
g=GetFolderItem("HardDisk:target folder")
f=GetFolderItem("StartUpVol:Release Notes")
If f <> Nil and g <> Nil then
 f.MoveFileTo(g)
 MsgBox "success!"
else
 MsgBox "Files not found"
end if
```

The following example creates a text file, changes the Creator from the default creator of "R*ch" to "ttxt", and writes some data to the file.

```
Dim f as FolderItem
Dim stream as TextOutputStream
f=GetSaveFolderItem("TEXT","Daily Planet Staff")
stream=f.CreateTextFile
f.MacCreator="ttxt"
Stream.WriteLine ("Perry White")
Stream.WriteLine ("Lois Lane")
Stream.WriteLine ("Jimmy Olsen")
Stream.Close
```

This example displays an open-file dialog box that lets the user select a QuickTime movie. The QuickTime movie is then copied into the Movie property of a MoviePlayer control.

```
Dim f As FolderItem
f=GetOpenFolderItem("video/quicktime")
If f<>Nil then
  MoviePlayer1.movie=f.OpenAsMovie
End if
```

This example copies all the files in a particular folder. The following code is a button's Action:

```
Dim origin, destination as FolderItem
origin=SelectFolder
If origin <> Nil then
 destination=SelectFolder
 If destination <> Nil then
  CopyFileOrFolder origin, destination
  MsgBox "Copy complete!"
 end if
end if
```

The CopyFileorFolder method is as follows:

```
Sub CopyFileorFolder (source as FolderItem, destination as FolderItem)
 Dim i as Integer
 Dim newFolder as FolderItem

 If source.directory then //it's a folder
  newFolder=destination.child(source.name)
  newFolder.createAsFolder
  For i=1 to source.count //go through each item
    If source.item(i).directory then
    //it's a folder
     CopyFileOrFolder source.item(i), newFolder
     //recursively call this
     //routine passing it the folder
    else
     source.item(i).CopyFileTo newFolder
    //it's a file so copy it
   end if
  next
 else //it's not a folder
  source.CopyFileTo destination
 end if
End Sub
```

Using GetRelative

```
Dim s as String
Dim f,g as FolderItem
f=new Folderitem
g=f.GetRelative(f.GetSaveInfo(Volume(0).Child("Documents"),0))
Statictext2.text=g.Absolutepath
```

See Also

GetFolderItem, GetOpenFolderItem, GetSaveFolderItem Volume, VolumeCount functions; BinaryStream, TextInputStream, TextOutputStream classes.

FolderItemDialog Class

Base class for the OpenDialog, SaveAsDialog, and SelectFolderDialog classes, which allow you to create customized open, save, and select folder dialog boxes.

Super Class Object

Properties

Name	Type	Description
ActionButtonCaption	String	Text of label for the Action button (e.g., Choose, Save, Open, etc., depending on context). Not necessarily the default button for the dialog.
CancelButtonCaption	String	Text of label for the Cancel button.
Filter	String	One or more File Types, separated by semicolons, previously defined in the File Types dialog box. *Filter* controls which files within *InitialDirectory* are visible.
InitialDirectory	FolderItem	Full or relative path to directory whose contents are displayed when the dialog first appears. *Filter* controls which files within this directory are visible.
Left	Integer	Distance (pixels) of the left side of the dialog from the left side of the main screen.
PromptText	String	Help text that appears within the dialog. See illustrations for OpenDialog, SaveAsDialog, and SelectFolderDialog.
Result	FolderItem	Holds the result of calling ShowModal() or ShowModalWithin. If the user validates the dialog, *Result* contains the FolderItem corresponding to the selection; otherwise, *Result* is Nil.
SuggestedFileName	String	Default name of the file; appears as the default text in the filename enterable area.
Title	String	String that appears in the Title bar.
Top	Integer	Distance (pixels) of the top of the dialog from the top of the main screen.

Methods

Name	Parameters	Description
ShowModal()		Opens the **FolderItemDialog**. Returns a Folderitem in *Result* if the user validates the dialog; otherwise *Result* is Nil.
ShowModalWithin	Parent as Window	Displays the **FolderItemDialog** as a Sheet window on Mac OS X within the window specified by *Parent*. If the application is not running under Mac OS X, ShowModalWithin is equivalent to ShowModal.

Notes

Macintosh versions of the **FolderItemDialog** dialogs require Navigation Services for nearly all the properties. Not all properties are available for all three subclasses of **FolderItemDialog** and some properties are not supported on Windows. The descriptions for each of the three subclasses have illustrations of the dialogs on each of the three platforms (Mac OS X, pre-Mac OS X with Navigation Services, and Windows 32).

The following table clarifies the situation.

Property	Macintosh			Windows		
	Open Dialog	SaveAs Dialog	Select Folder	Open Dialog	SaveAs Dialog	Select Folder
Top & Left	✔	✔	✔	✔	✔	✔
PromptText	✔	✔	✔			✔
Title	✔	✔	✔	✔	✔	✔
ActionButtonCaption	✔	✔	✔			✔
CancelButtonCaption	✔	✔	✔			
SuggestedFileName		✔		✔	✔	
Filter	✔			✔	✔	
InitialDirectory	✔	✔	✔	✔	✔	✔

If Navigation Services is not installed on pre-Mac OS X machines, only a very limited subset of the functionality shown in the above table is actually available. For OpenDialog, only the Filter property is supported; for SaveAsDialog, only the PromptText and SuggestedFileName properties are supported. Navigation Services is installed by default on systems 8.5–9.1, but can be installed manually on earlier systems.

Examples

See the examples and illustrations for the OpenDialog, SaveAsDialog, and SelectFolderDialog classes.

See Also

OpenDialog, SaveAsDialog, SelectFolderDialog classes.

Font Function

Used to access the names of fonts installed on the user's computer.

Syntax

result*=Font*(index)

Part	Type	Description
result	String	The name of the font whose index number is passed.
index	Integer	The number of the font.

Notes

Fonts are accessed in alphabetical order where font 0 is the first font. Use the FontCount function to determine the number of fonts.

Examples

This is an example of a function that determines if the font named passed is installed on the user's computer:

```
Function FontAvailable(FontName As String) as Boolean
Dim i,n as Integer
n=FontCount-1
  For i=0 to n
    If Font(i)=FontName Then
      Return True
    End If
  Next
  Return False
End Function
```

This example dynamically creates a Font menu using a menu item array named FontFontName:

```
Dim m as MenuItem
Dim i,n as Integer
n= FontCount-1
FontFontName(0).Text=Font(0)
For i=1 to n
  m=New FontFontName
  m.Text=Font(i)
Next
```

See Also

FontCount, FontsFolder functions.

FontCount Function

Used to determine the number of fonts installed on the user's computer.

Syntax

***result*=FontCount**

Part	Type	Name
result	Integer	The number of fonts installed.

Notes

FontCount is useful when you need to build a list of available fonts or need to determine if a specific font is installed.

Examples

See the example for the Font function.

See Also

Font, FontsFolder functions.

FontsFolder Function

Used to access the Fonts folder in the System folder. On Windows, it accesses the WINDOWS/FONTS directory.

Syntax

***result*=FontsFolder**

Part	Type	Description
result	FolderItem	A FolderItem that represents the Fonts folder.

Notes

Use the **FontsFolder** function to access the Fonts folder.

The **FontsFolder** function provides a way to access the Fonts folder that will work under different language systems, and will continue to work even if Apple reorganizes the System folder. If your application will run on Windows, you can check the TargetWin32 global constant before calling **FontsFolder**.

If you are looking for a way to create a list of fonts the user has installed, see the FontCount and Font functions.

Examples

This example places all the names of the items in the Fonts folder in a ListBox.

```
Dim i as Integer
Dim f as FolderItem
f=FontsFolder
For i=1 to f.count
  ListBox1.addrow f.item(i).name
Next
```

See Also

AppleMenuFolder, ApplicationSupportFolder, ControlPanelsFolder, DesktopFolder, ExtensionsFolder, Font, FontCount, PreferencesFolder, StartupItemsFolder, SystemFolder, TemporaryFolder, TrashFolder functions.

For Each...Next Statement

Loops through the elements of a one-dimensional array.

Syntax

For Each *element* In *array*

[statements]

Next

Part	Description
element	A variable of the same data type as *array* that refers to an element of *array*. The loop processes each value in *array*.
array	A one-dimensional Integer, Single, Double, or String array.

Examples

The following function adds up the numbers in the array passed to it.

```
Function SumStuff(values() as Double) as Double
 Dim sum, element as Double
 For Each element In values
  sum=sum+element
 Next
Return Sum
```

To call the function, pass an array of doubles in a statement such as:

```
s=SumStuff(MyNumbers)
```

where s is declared as a Double and MyNumbers is an array of Doubles.

See Also

For...Next statement, Dim statement.

For...Next Statement

Executes a series of statements a specified number of times. The For Each…Next statement is a variation that executes once for each element of a one-dimensional array.

Syntax

For loops have two syntaxes:

For *counter* = *start* To *end* [Step *step*]

[Statements]

[Exit]

[Statements]

Next

or

For *counter* =*start* DownTo *end* [Step *step*]

[Statements]

{Exit}

[Statements]

Next

Part	Description
counter	A local variable that will be incremented each time the loop executes. It can be an Integer, Single, or Double.
start	Initial value of *counter*.
end	Final value of *counter*.
step	Amount to increment *counter* each time *statements* are executed. *Step* is 1 unless otherwise specified.
statements	Statements to be executed the specified number of times (until *counter* is greater than *end*, or less than *end* if DownTo is used).

Notes

If *start*, *end, or step* is an expression that must be evaluated, the For loop does the evaluation each time the loop is executed, even if the expression evaluates to the same value for every iteration. Therefore, the loop:

```
For i=0 to FontCount-1
.
.
Next
```

will run more slowly than coding:

```
Dim nFonts as Integer
nFonts=FontCount-1
For i=0 to nFonts
.
.
Next
```

The size of the difference, of course, depends on the speed of the computer and the time required to evaluate the expression.

You can either increment or decrement the counter. If *step* is positive or zero, the loop executes until *counter* is greater than *end*.

To decrement the counter, use the second syntax using **DownTo** and a positive value of *step*.

Counter is incremented or decremented when the **Next** statement is reached.

Counter must be a local variable and must be an integer.

The Exit statement can be used to force the loop to stop prematurely. The Exit statement causes control to resume at the statement that follows **Next**.

For...Next statements can be nested to any level. However, each **For...Next** statement must have a different *counter*.

When a For loop runs, it takes over the interface, preventing the user from interacting with menus and controls. Ordinarily, this is of no concern. If, however, the loop is very lengthy, you can move the code for the loop to a separate Thread, allowing it to execute in the background.

Examples

This example uses the **For...Next** statement to test the values in an array and change those that are "Today" to "Tomorrow".

```
Dim i,last As Integer
last=Ubound(aDays)
For i=1 to last
  If aDays(i)="Today" Then
    aDays(i)="Tomorrow"
  End If
Next
```

The following example uses the DownTo keyword to decrement the counter:

```
Dim i as Integer
For i=5 DownTo 1 Step 1
 Beep
Next
```

See Also

Do...Loop, While...Wend, For…Each statements.

Format Function

Returns as a string a formatted version of the number passed based on the parameters specified. The **Format** function is similar to the way spreadsheet applications format numbers.

Syntax

***result*=Format(*number*, *formatSpec*)**

Part	Type	Description
result	String	The formatted number.
number	Double	The number to be formatted.
formatSpec	String	Defines the formatting to be applied to the number passed.

Notes

The *formatSpec* is a string made up of one or more special characters that control how the number will be formatted:

Character	Description
#	Placeholder that displays the digit from the value if it is present.
0	Placeholder that displays the digit from the value if it is present. If no digit is present, 0 (zero) is displayed in its place.
.	Placeholder for the position of the decimal point.
,	Placeholder that indicates that the number should be formatted with thousands separators.

Character	Description
%	Displays the number multiplied by 100.
(	Displays an open paren.
)	Displays a closing paren.
+	Displays the plus sign to the left of the number if the number is positive or a minus sign if the number is negative.
-	Displays a minus sign to the left of the number if the number is negative. There is no effect for positive numbers.
E or e	Displays the number in scientific notation.
\character	Displays the character that follows the backslash.

The absolute value of the number is displayed. You must use the + or - signs if you want the sign displayed.

Although the special formatting characters are U.S. characters, the actual characters that will appear are based on the settings in the user's Numbers Control Panel.

The *formatSpec* can be made up of up to three formats separated by semi-colons. The first format is the format to be used for positive numbers. The second format is the format to be used for negative numbers and the third format is the format to be used for zero.

Examples

The following are several examples that use the various special formatting characters.

Format	Number	Formatted String
#.##	1.784	1.78
#.0000	1.3	1.3000
0000	5	0005
#%	0.25	25%
###,###.##	145678.5	145,678.5
#.##e+	145678.5	146e+5
-#.##	-3.7	-3.7
+#.##	3.7	+3.7
#.##;(#.##);\z\e\r\o	3.7	3.7
#.##;(#.##);\z\e\r\o	-3.7	(3.7)
#.##;(#.##);\z\e\r\o	0	zero

The following example returns the number 3560.3 formatted as $3,560.30.

```
Dim s as String
s=Format(3560.3, "\$###,##0.00")
```

FrameColor Function

The currently selected Appearance Manager color for outlining objects.

Syntax

***result*=FrameColor**

Part	Type	Description
result	Color	The color used for drawing the outline of a control.

Notes

This value is useful when you are using Canvas controls to create custom controls that you want to be Appearance Manager savvy. When drawing objects, use this color for the object's frame. ListBoxes, for example, use this color to draw the dark frame around the ListBox.

This value can be changed by the user on the fly using the Kaleidoscope Control Panel, so you should access this value during your Paint Event handler rather than storing the value.

See Also

DarkBevelColor, DarkTingeColor, FillColor, HighlightColor, LightBevelColor, LightTingeColor, TextColor functions; Color data type.

Function Statement

Declares the name, parameters, returned value, and code that form the body of a function (method).

Syntax

Function *name*([(*parameterList*)]) As *type*

[local variable declarations]

[statements]

[Return]

[statements]

[exception handlers]

[Finally]

End Function

Part	Description
name	Required. The name of the function (method); follows standard variable naming conventions.
parameterList	Optional. List of values representing parameters that are passed to the function when it is called. Multiple parameters are separated by commas.
type	The data type of the value returned by the function.

Notes

The **Function** statement is used to define a method that can be used on the right side of an expression just like the built-in functions such as Abs, Len, etc. Methods are always associated with an object. All executable code must be in a Sub or **Function** statement. A **Function** differs from a Sub in that a **Function** can be used on the right side of an expression and a Sub cannot. A function can not be defined inside another Sub or function. A function executes each line of code from the top down. Once the last line of code is executed, control returns to the line that called the function.

You call a function by using its name followed by its parameters in parentheses.

All variables used in a function must be declared before they are used in a statement. Variables and parameters used in a function are local. Properties can be accessed from within any Sub or function. Global properties are created by calling the Dim statement within the Sub of a module. Local variables are variables that are created each time the function is run and destroyed when the function finishes. Consequently, they can only be accessed by the statements within the function. They are created by using the Dim

statement from within a Sub or function except those attached to a module (as that would make them global variables).

The **Return** statement can be used to immediately return control to the statement that called the function and to pass back a value to the left side of the statement. If the **Return** statement is not called, the null value of the data type returned by the function is returned.

Exception handlers are statements that handle errors. See the RuntimeException class and the Exception Block for more information.

If your method does not need to return a value, you will want to declare it as a Sub. A Sub is a method that does not return a value. See the Sub statement for more information.

If you need to return several values, you can use the ByRef keyword when you define the routine's parameters. Using ByRef, you can return the results into the parameters.

A **Function** method can call itself, resulting in recursion. Too much recursion can lead to stack overflow errors.

Sometimes a function needs to do some cleanup work whether it is finishing normally or aborting early because of an exception. The optional "**Finally**" block at the end of the function serves this purpose. Code in this block will be executed even if an exception has occured, whether the exception was handled or not.

Examples

This example is a function that calculates area based on the length and height passed.

```
Function Area(Length as Double, Height as Double) As Double
  Dim theArea As Double
  theArea=Length*Height
  Return theArea
End Function
```

This example shows the Area function above written in a simpler form (without the extra local variable).

```
Function Area(Length as Double, Height as Double) As Double
  Return Length*Height
End Function
```

This example shows the Area function above being called and its returned value assigned to variable.

```
Dim a As Double
a=Area(10,10) //returns 100
```

See Also

Exit statement, Raise statement, RuntimeException class, Sub statement.

Examples

This code has several examples of ways in which this error can be generated.

```
Dim d as Double
Dim c as Color
Atan2(1,0) //not assigned to anything
RGB(100,100,100) //incorrect

d=Atan2(1,0) //correct
c=RGB(100,100,100) //correct
```

GetAppleEventTarget Function

Used to bring up a dialog box that allows the user to choose the target application on his computer or a networked computer. The dialog box shows computers on the left and applications on the right. The applications list is labelled with the parameter *ListLabel*.

Syntax

***result*=GetAppleEventTarget (*Prompt*, *ListLabel*)**

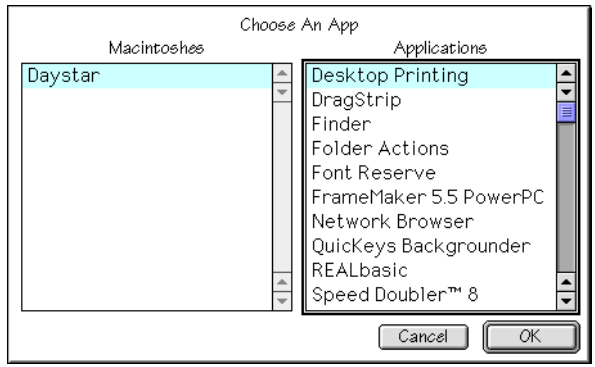
Part	Type	Description
Result	AppleEventTarget	Reference to target application
Prompt	String	Prompt text that will appear as the header of the dialog box.
ListLabel	String	Label for the list of applications in the dialog box.

Notes

This function works under Mac OS "classic" only. In particular, it does not work under Carbon or Mac OS X.

Example

The following example displays the Choose an App dialog box shown below.




```
Dim AE as AppleEventTarget
AE=GetAppleEventTarget("Choose An App","Applications"
```

GetFolderItem Function

Used to access an item in a folder (directory) using a path.

Syntax

***result*=GetFolderItem(*path*)**

Part	Type	Description
result	FolderItem	Represents the file that was opened.
path	String	Full path to the item or string from FolderItem's GetSaveInfo method.

Notes

The **GetFolderItem** function creates a FolderItem object for a specific item (application, document, or folder). **GetFolderItem** can be passed a full path to a specific item or a string generated by the GetSaveInfo method of the FolderItem class. The string should contain absolute path information. If it contains only relative path information, it is ignored.

If *path* is invalid, **GetFolderItem** returns Nil. An invalid path contains the name of a volume or directory that does not exist. If *path* contains a valid volume name (and, optionally, valid directory names) but the name of a nonexistent file, then **GetFolderItem** will return a FolderItem that refers to the filename that was passed.

When passing absolute pathnames to **GetFolderItem**, you may need to take into account the fact that Mac OS and Windows use different separator characters in

pathnames. If your application will run on multiple operating systems, see the note on specifying pathnames in the Notes section for the FolderItem class.

If you are intending to open an existing file with **GetFolderItem**, you should check the Exists property of the FolderItem to be sure that the file actually exists before accessing that FolderItem's properties.

If you are unsure that *path* is valid, you should include an Exception block that handles Nil FolderItems or use an If statement to check that the FolderItem is not Nil before trying to access the FolderItem. This is not always the case. If *path* contains, for example, only the name of a nonexistent file, then the FolderItem will never be Nil.

Passing an empty string returns a FolderItem representing the folder the project is in. If you haven't saved the project, it returns the folder the REALbasic application is in.

GetFolderItem automatically resolves aliases when *path* represents an alias. To prevent this, use GetTrueFolderItem.

Examples

This example displays the name of the folder that contains the REALbasic project in a message box.

```
Dim currentFolder as FolderItem
currentFolder=GetFolderItem("")
MsgBox currentFolder.name
```

The following example opens a TIFF file in the same folder as the application (or the same folder as REALbasic if run from the IDE) and uses it as the background image ("backdrop") for a Canvas control:

```
Dim f as FolderItem
f=GetFolderItem("Zippy.tif")
If f.exists then
 Canvas1.backdrop=f.OpenAsPicture
end if
```

This example attempts to open a PICT document and, if it exists, displays it in an ImageWell. It checks for both an invalid path and a nonexistent file. If the path is invalid, a NilObjectException is generated when **GetFolderItem** tries to return a

FolderItem. The NilObjectException is handled by the Exception block at the end of the method.

```
Dim f as FolderItem
f= GetFolderItem("HD:Images:Logo")
If Not f.exists then
 Beep
 MsgBox "The file "+f.absolutePath+"doesn't exist!"
else //document exists
 If f.MacType="PICT" then
   imageWell1.image=f.OpenAsPicture
 end if
end if
Exception err as NilObjectException
MsgBox "Invalid pathname!"
```

See Also FolderItem, FolderItemDialog classes, GetTrueFolderItem function, RuntimeException class, NilObjectException error, Nil object.

GetFontTextEncoding Function

Used to get the text encoding to be used as the output encoding when you are going to display the text in that font. Most fonts will have the same encoding but fonts like Symbol have a different encoding.

Syntax ***result*=GetFontTextEncoding(*FontName*)**

Part	Type	Description
result	TextEncoding	Text encoding of *FontName*
FontName	String	Name of a font. It returns the system default encoding if you pass one of the metafont names (e.g., "System", "SmallSystem", "System.UTF8", etc.).

Example The following code returns the text encoding for the Symbol font: Base=33 (MacSymbol), Variant=0, Internet Name=X-MAC-SYMBOL.

```
Dim t as TextEncoding
t=GetFontTextEncoding("Symbol")
```

See Also TextEncoding class.

GetIndexedObjectDescriptor Function

Returns an AppleEventObjectSpecifier that is within another AppleEventObjectSpecifier. The AppleEventObjectSpecifier is found by class and index.

Syntax

result*=GetIndexedObjectDescriptor*(DesiredClass, Object, index)

Part	Type	Description
result	AppleEventObjectSpecifier	The object that was found.
DesiredClass	String	Indicates the class of AppleEvent object you are looking for.
Object	AppleEventObjectSpecifier	The source object. Pass Nil to search at the application level.
Index	Integer	The number of the object you want (starting at 1).

See Also

AppleEvent class, GetNamedObjectDescriptor, GetOrdinalObjectDescriptor, GetPropertyObjectDescriptor, GetRangeObjectDescriptor, GetStringComparisonObjectDescriptor, GetTestObjectDescriptor functions.

GetInternetTextEncoding Function

When decoding mail or web pages you would use this function to get the text encoding rather than GetTextEncoding. This function is the inverse of TextEncoding.InternetName. It returns the encoding associated with a standard name for that encoding as used on the Internet.

Syntax

***result*=GetInternetTextEncoding(*InternetEncoding*)**

Part	Type	Description
result	TextEncoding	Text encoding of *InternetEncoding*
InternetEncoding	String	Internet text.

See Also

GetTextEncoding, TextEncoding functions.

GetNamedObjectDescriptor Function

Returns an AppleEventObjectSpecifier that is within another AppleEventObjectSpecifier. The AppleEventObjectSpecifier is found by class and name.

Syntax

result*=GetNamedObjectDescriptor*(DesiredClass, Object, Name)

Part	Type	Description
result	AppleEventObjectSpecifier	The object that was found.
DesiredClass	String	Indicates the class of AppleEvent object you are looking for.
Object	AppleEventObjectSpecifier	The source object. Pass Nil to search at the application level.
Name	String	The name of the object you are looking for.

See Also

AppleEvent class, GetIndexedObjectDescriptor, GetOrdinalObjectDescriptor, GetPropertyObjectDescriptor, GetRangeObjectDescriptor, GetStringComparisonObjectDescriptor, GetTestObjectDescriptor function.

GetOpenFolderItem Function

Used to access a document or application selected by the user via the standard open-file dialog box.

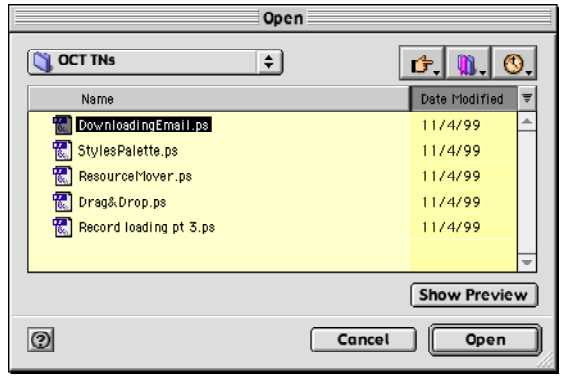
Syntax

***result*=GetOpenFolderItem(*filter*)**

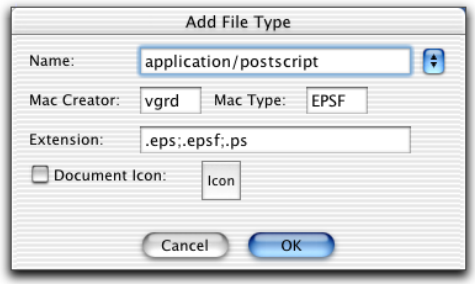
Part	Type	Description
result	FolderItem	Represents the file that was opened by the user. *Result* will be Nil if the user clicked the Cancel button.
filter	String	Semi-colon separated list of file types the user can open. The file types must be defined in the File Types dialog box.

Notes

The **GetOpenFolderItem** function displays the standard open-file dialog box. If the user has Mac OS system 8.5 (or above) installed, the new Navigation Services open-file dialog box appears as a movable modal dialog.



The *filter* parameter is used to limit the types of files that the user can open to one or more of the file types defined in the File Types dialog box. The *filter* is a semi-colon separated list of file type names. For example, if you wanted the user to be able to open only text files and postscript files when making a particular call to the **GetOpenFolderItem** function, you would add two file types to the File Types dialog box. Using the built-in file type templates, you would add the "application/text" type and the "application/postscript" type. This is shown below. Note that Windows extensions are filled in automatically when you choose from the File Types pop-up.



You would then pass "application/text; application/postscript" as the filter to the **GetOpenFolderItem** function.

Only files whose type matches one of the file types passed in the *filter* will be displayed in the open file dialog box. If you want to display all files, you will need to add a file type to the File Types dialog box that uses "????" as its Macintosh Type. The built-in file type, special/any, specifies that any file type is acceptable.

The **GetOpenFolderItem** function returns a FolderItem that represents the file the user selected. You can then use the FolderItem to access various data about the file such as its name, full path, etc. See the FolderItem class for more information.

If the user clicks the Cancel button in the open file dialog box, the FolderItem will be Nil. You can test for this by comparing the FolderItem with the Nil value. Accessing a Nil FolderItem will cause an error.

Examples

This example displays an open-file dialog box that lets the user select a QuickTime movie. The QuickTime movie is then copied into the Movie property of a MoviePlayer control.

```
Dim f As FolderItem
f=GetOpenFolderItem("video/quicktime")
If f<>Nil then
  MoviePlayer1.movie=f.OpenAsMovie
End if
```

See Also

GetSaveFolderItem, SelectFolder functions; FolderItemDialog class.

GetOrdinalObjectDescriptor Function

Returns an AppleEventObjectSpecifier that is within another AppleEventObjectSpecifier. The AppleEventObjectSpecifier is found by class and an ordinal key.

Syntax

result*=GetOrdinalObjectDescriptor*(DesiredClass, Object, OrdinalKey)

Part	Type	Description
result	AppleEventObjectSpecifier	The object that was found.
DesiredClass	String	Indicates the class of AppleEvent object you are looking for.
Object	AppleEventObjectSpecifier	The source object. Pass Nil to search at the application level.
OrdinalKey	String	The scope to use when searching.

Notes

The ordinal keys are:

Key	Description
firs	First
last	Last
midd	Middle

Key	Description
any	Any
all	All

Ordinal keys are four characters so the “any” and “all” keys have a space following them.

Example

This example asks the Finder for a count of the currently running processes:

```
Dim a as AppleEvent
Dim i, count as Integer
a = NewAppleEvent("core", "cnte", "MACS")
a.MacTypeParam("kocl") = "prcs"
a.ObjectSpecifierParam("----") = GetOrdinalObjectDescriptor("prcs", nil, "all ")
If a.Send then
  count = Val(a.ReplyString)
End if
```

See Also

AppleEvent class, GetIndexedObjectDescriptor, GetNamedObjectDescriptor, GetPropertyObjectDescriptor, GetRangeObjectDescriptor, GetStringComparisonObjectDescriptor, GetTestObjectDescriptor function.

GetPropertyObjectDescriptor Function

Returns an AppleEventObjectSpecifier that refers to a property of the AppleEventObjectSpecifier passed.

Syntax

result*=GetPropertyObjectDescriptor*(Object, Name)

Part	Type	Description
result	AppleEventObjectSpecifier	An object that refers to the Name property of *Object*.
Object	AppleEventObjectSpecifier	The source object. Pass Nil to search at the application level.
Name	String	The name of the property of *Object* to be returned.

See Also

AppleEvent class, GetIndexedObjectDescriptor, GetNamedObjectDescriptor, GetOrdinalObjectDescriptor, GetRangeObjectDescriptor, GetStringComparisonObjectDescriptor, GetTestObjectDescriptor function.

GetQTCrossFadeEffect Function

Creates a cross-fade effect. Use QTEffectSequence to create an effect sequence using the cross-fade effect.

Syntax

***result*=GetQTCrossFadeEffect**

Part	Type	Description
result	QTEffect	Cross-fade effect

Example

This example creates a QuickTime movie from two pictures, p1 and p2, using the cross-fade effect.

```
Dim theEffect as QTEffect
Dim sequence as QTEffectSequence
theEffect=GetQTCrossFadeEffect
sequence=New QTEffectSequence(theEffect,p1,p2,96)
```

See Also

QTTrack, QTVideoTrack, QTEffect, QTEffectSequence classes; GetQTSMPTEEffect function.

GetQTGraphicsExporter Function

Creates an exporter object that uses a specified graphics format.

Syntax

***result*=GetQTGraphicsExporter(*Format*)**

Part	Type	Description
result	QTGraphicsExporter	Exporter object.
Format	String	Format used to save the graphic. BMPf - BMP file format PNGf - PNG file format PNTG - MacPaint 8BPS - Photoshop .SGI - SGI TPIC - Targa JPEG - Jpeg

Example

See the example for QTGraphicsExporter.

See Also

QTGraphicsExporter class.

GetQTSMPTEEffect Function

Gets an SMPTEE effect.

Syntax

***result*=GetQTSMPTEEffect(*ID*)**

Part	Type	Description
result	QTEffect	The selected SMPTE effect.
ID	Integer	ID of SMPTE effect (see the following table)

Table 2: SMPTE Effects

ID	Type	Effect
1	Wipe	Slide horizontal
2	Wipe	Slide vertical
3	Wipe	Top left
4	Wipe	Top right
5	Wipe	Bottom right
6	Wipe	Bottom left
7	Wipe	Four corner
8	Wipe	Four box
21	Wipe	Barn vertical
22	Wipe	Barn horizontal
23	Wipe	Top center
24	Wipe	Right center
25	Wipe	Bottom center
26	Wipe	Left center
41	Wipe	Diagonal left down
42	Wipe	Diagonal right down
43	Wipe	Vertical bow tie
44	Wipe	Horizontal bow tie
45	Wipe	Diagonal left out
46	Wipe	Diagonal right out
47	Wipe	Diagonal cross
48	Wipe	Diagonal box
61	Wipe	Filled V
62	Wipe	Filled V right
63	Wipe	Filled V bottom

Table 2: SMPTE Effects (Continued)

ID	Type	Effect
64	Wipe	Filled V left
65	Wipe	Hollow V
66	Wipe	Hollow V right
67	Wipe	Hollow V bottom
68	Wipe	Hollow V left
71	Wipe	Vertical zig zag
72	Wipe	Horizontal zig zag
73	Wipe	Vertical barn zig zag
74	Wipe	Horizontal barn zig zag
501	Wipe	Random wipe — one of the 34 SMPTE wipe effects is chosen at random
101	Iris	Rectangle
102	Iris	Diamond
103	Iris	Triangle
104	Iris	Triangle right
105	Iris	Triangle upside down
106	Iris	Triangle left
107	Iris	Arrowhead
108	Iris	Arrowhead right
109	Iris	Arrowhead upside down
110	Iris	Arrowhead left
111	Iris	Pentagon
112	Iris	Pentagon upside down
113	Iris	Hexagon
114	Iris	Hexagon side
119	Iris	Circle
120	Iris	Oval
121	Iris	Oval side
122	Iris	Cat eye
123	Iris	Cat eye slide
124	Iris	Round rect
125	Iris	Round rect side
127	Iris	4 point star
128	Iris	5 point star
129	Iris	6 point star
130	Iris	Heart

Table 2: SMPTE Effects (Continued)

ID	Type	Effect
131	Iris	Keyhole
502	Iris	Random iris — one of the 26 SMPTE iris effects is chosen at random
201	Radial	Rotating top
202	Radial	Rotating right
203	Radial	Rotating bottom
204	Radial	Rotating left
205	Radial	Rotating top bottom
206	Radial	Rotating left right
207	Radial	Rotating quadrant
211	Radial	Top to bottom 180°
212	Radial	Right to left 180°
213	Radial	Top to bottom 90°
214	Radial	Right to left 90°
221	Radial	Top 180°
222	Radial	Right 180°
223	Radial	Bottom 180°
224	Radial	Left 180°
225	Radial	Counter rotating top bottom
226	Radial	Counter rotating left right
227	Radial	Double rotating top bottom
228	Radial	Double rotating left right
231	Radial	V Open top
232	Radial	V Open right
233	Radial	V Open bottom
234	Radial	V Open left
235	Radial	V Open top bottom
236	Radial	V Open left right
241	Radial	Rotating top left
242	Radial	Rotating bottom left
243	Radial	Rotating bottom right
244	Radial	Rotating top right
245	Radial	Rotating top left bottom right
246	Radial	Rotating bottom left top right
251	Radial	Rotating top left right
252	Radial	Rotating left top bottom

Table 2: SMPTE Effects (Continued)

ID	Type	Effect
253	Radial	Rotating bottom left right
254	Radial	Rotating right top bottom
261	Radial	Rotating double center right
262	Radial	Rotating double center top
263	Radial	Rotating double center top bottom
264	Radial	Rotating double center left right
503	Radial	Random radial — one of the 39 SMPTE radial effects is chosen at random
301	Matrix	Horizontal matrix
302	Matrix	Vertical matrix
303	Matrix	Top left diagonal matrix
304	Matrix	Top right diagonal matrix
305	Matrix	Bottom right diagonal matrix
306	Matrix	Bottom left diagonal matrix
310	Matrix	Clockwise top left matrix
311	Matrix	Clockwise top right matrix
312	Matrix	Clockwise bottom right matrix
313	Matrix	Clockwise bottom left matrix
314	Matrix	Counterclockwise top left matrix
315	Matrix	Counterclockwise top right matrix
316	Matrix	Counterclockwise bottom right matrix
317	Matrix	Counterclockwise bottom left matrix
320	Matrix	Vertical start top matrix
321	Matrix	Vertical start bottom matrix
322	Matrix	Vertical start top opposite matrix
323	Matrix	Vertical start left matrix
324	Matrix	Horizontal start left matrix
325	Matrix	Horizontal start right matrix
326	Matrix	Horizontal start left opposite
327	Matrix	Horizontal start right opposite matrix
328	Matrix	Double diagonal top right matrix
329	Matrix	Double diagonal bottom right matrix
340	Matrix	Double spiral top matrix
341	Matrix	Double spiral bottom matrix
342	Matrix	Double spiral left matrix
343	Matrix	Double spiral right matrix

Table 2: SMPTE Effects (Continued)

ID	Type	Effect
344	Matrix	Quad spiral vertical matrix
345	Matrix	Quad spiral horizontal matrix
350	Matrix	Vertical waterfall left matrix
351	Matrix	Vertical waterfall right matrix
352	Matrix	Horizontal waterfall left matrix
353	Matrix	Horizontal waterfall right matrix
504	Matrix	Random matrix — one of the 34 SMPTE matrix effects is chosen at random
409		Random effect — one of the 133 SMPTE effects is chosen at random

Example

See the example for QTVideoTrack.

See Also

GetQTCrossFadeEffect function, QTEffect, QTEffectSequence, QTTrack, QTVideoTrack classes.

GetRangeObjectDescriptor Function

Returns an AppleEventObjectSpecifier containing a range of objects contained in the AppleEventObjectSpecifier passed.

Syntax

result*=GetRangeObjectDescriptor*(DesiredClass, Object, RangeStart, RangeEnd)

Part	Type	Description
result	AppleEventObjectSpecifier	The object containing the range of objects specified.
DesiredClass	String	The class of AppleEvent object you are looking for.
Object	AppleEventObjectSpecifier	The source object. Pass Nil to search at the application level.
RangeStart	AppleEventObjectSpecifier	Indicates the start of the range.
RangeEnd	AppleEventObjectSpecifier	Indicates the end of the range.

See Also

AppleEvent class, GetIndexedObjectDescriptor, GetNamedObjectDescriptor, GetOrdinalObjectDescriptor, GetPropertyObjectDescriptor, GetStringComparisonObjectDescriptor, GetTestObjectDescriptor function.

GetSaveFolderItem Function

Used to present the standard Save As dialog box to the user.

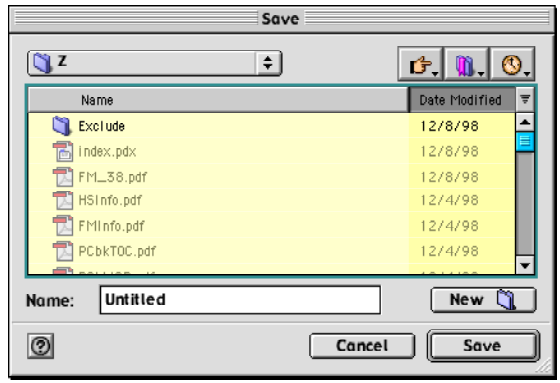
Syntax

***result*=GetSaveFolderItem(*filter*, *default file name*)**

Part	Type	Description
result	FolderItem	Represents the file the user wants to create.
filter	String	The file type to be assigned to the file, as defined in the File Types dialog box.
default file name	String	The name that should appear by default in the Save As dialog box.

Notes

The **GetSaveFolderItem** function displays the standard Save As file dialog box, allowing the user to choose a location and enter a name for the file to be saved. If the user has Mac OS System 8.5–9.1 installed, the new Navigation Services save-file dialog box is displayed automatically.



The **GetSaveFolderItem** function does not create the file. It simply returns a FolderItem that represents the potential file. To create the actual file, you will need to call the CreateBinaryFile or CreateTextfile method for the FolderItem.

The filter should either be an empty string or the name of a file type as defined in the File Types dialog box, such as “image/gif”, ”image/jpeg”, or “application/rtf”.

On Mac OS X, a Hide Filename Extension checkbox appears in the save-file dialog. The FolderItem returned has its ExtensionVisible property set according to the user's use of this checkbox.

Examples

This example displays the save as file dialog box. A text file is then created and the text properties of three EditFields are written to the new file. Finally the file is closed.

```
Dim file As FolderItem
Dim fileStream as TextOutputStream
file=GetSaveFolderItem("","My Info")
If file<>Nil then
  fileStream=file.CreateTextFile
  fileStream.WriteLine namefield.text
  fileStream.WriteLine addressfield.text
  fileStream.WriteLine phonefield.text
  fileStream.Close
end if
```

See Also

GetOpenFolderItem, SelectFolder functions; FolderItem, FolderItemDialog classes.

GetStringComparisonObjectDescriptor Function

Returns the AppleEventObjectSpecifier that matches the string comparison passed.

Syntax

result*=GetStringComparisonObjectDescriptor *(ComparisonKey, ComparisonForm, Field, Value)

Part	Type	Description
result	AppleEventObjectSpecifier	The object that was found.
ComparisonKey	String	The type of comparison to be performed. This string must be four characters in length.
ComparisonForm	String	The form of the comparison. - For example, “prop” indicates that the comparison will be checking a property.
Field	String	The field the comparison will be performed on - the ComparisonForm parameter controls how the field parameter is interpreted.
Value	String	The value that the comparison will be performed against.

Notes

Valid comparison keys are:

Comparison Key	Description
=	Equals
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to

Comparison keys must be four characters. You must include spaces after the characters in the comparison key so that the string passed is four characters in length.

See Also

AppleEvent class, GetIndexedObjectDescriptor, GetNamedObjectDescriptor, GetOrdinalObjectDescriptor, GetPropertyObjectDescriptor, GetRangeObjectDescriptor, GetTestObjectDescriptor functions.

GetTemporaryFolderItem Function

The **GetTemporaryFolderItem** function creates a FolderItem object in the current Temporary Folder.

Syntax

***result*=GetTemporaryFolderItem**

Part	Type	Description
result	FolderItem	FolderItem object that refers to a document in the Temporary Folder.

Example

The following code creates FolderItem in the active Temporary folder and displays its absolute pathname.

```
Dim f as FolderItem
f=GetTemporaryFolderItem
MsgBox f.absolutePath
```

See Also

FolderItem class, GetFolderItem, TemporaryFolder functions.

GetTestObjectDescriptor Function

Returns an AppleEventObjectSpecifier built to perform some kind of test.

Syntax

***result*=GetTestObjectDescriptor(*DesiredClass*, *Object*, *Comparison*)**

Part	Type	Description
result	AppleEventObjectSpecifier	The object that was created.
DesiredClass	String	The class of AppleEvent object you are looking for.
Object	AppleEventObjectSpecifier	The source object. Pass Nil to search at the application level.
Comparison	AppleEventObjectSpecifier	The object describing the actual test to be performed.

Notes

At present, GetStringComparisonObjectDescriptor is the only function that returns an AppleEventObjectSpecifier describing a test, but this will probably be extended in the future.

See Also

AppleEvent class, GetIndexedObjectDescriptor, GetNamedObjectDescriptor, GetOrdinalObjectDescriptor, GetPropertyObjectDescriptor, GetRangeObjectDescriptor, GetStringComparisonObjectDescriptor functions.

GetTextConverter Function

Creates a TextConverter object for converting between two encodings. It also works if the input encoding and output encoding are the same.

Syntax

***result*=GetTextConverter(*inputEncoding*, *outputEncoding*)**

Part	Type	Description
result	TextConverter	Object of type TextConverter, used to perform conversion from *InputEncoding* to *OutputEncoding*.
inputEncoding	TextEncoding	Input text encoding.
outputEncoding	TextEncoding	Output text encoding.

Notes

In REALbasic, every string may internally have a record of its encoding as well as the bytes that constitute its actual text. In many cases, the encoding is unknown, but any string returned by a TextConverter will have a known encoding, and so will be treated properly by EditFields and Graphics.DrawString. This is why it may sometimes be useful to get a TextConverter where the input and output encodings are the same; it provides a way to make sure that REALbasic knows what encoding a string represents.

Example

The following example converts the text in an EditField:

```
Dim c as TextConverter
c=GetTextConverter(GetTextEncoding(&h500), getTextEncoding(0))
EditField2.text=c.convert(EditField1.text)
```

See Also

GetTextEncoding function, TextConverter class, GetFontTextEncoding function, GetInternetTextEncoding function, Encodings object, Appendix A of the printed or pdf versions of the *Language Reference*.

GetTextEncoding Function

Returns a TextEncoding object based on user-specified parameters.

Syntax

***result*=GetTextEncoding (*Base*, *[Variant*, *Format]*)**

Part	Type	Description
result	TextEncoding	TextEncoding object.
Base	Integer	The type of encoding.
Variant	Integer	Specific variation of *Base*
Format	Integer	Format—only used by Unicode for defining which format of Unicode you wish to use

Notes

See "Text Encoding Base and Variant Codes" on page 591 in the pdf or print Language Reference for the codes for *Base* and *Variant*. The codes for *Format* are as follows:

Name	Code
Default Format	0
Unicode16BitFormat	0
UnicodeUTF7Format	1
UnicodeUTF8Format	2
Unicode32BitFormat	3

Example

See the example for GetTextConverter.

See Also

GetTextConverter, GetFontTextEncoding, GetInternetTextEncoding functions; Encodings object.

GetTrueFolderItem Function

Returns a FolderItem that refers the true path (without resolving any aliases) of a file or folder.

Syntax

***result*=GetTrueFolderItem(*Name*)**

Part	Type	Description
result	FolderItem	FolderItem that refers to Name.
Name	String	Pathname to file or folder.

Note

If the item referenced by *Name* does not exist, **GetTrueFolderItem** returns Nil.

Example

The following example compares **GetTrueFolderItem** with GetFolderItem. While both functions are passed the same absolute path to an alias, f refers to the alias, while g refers to the actual application.

```
Dim f, g as FolderItem
f=GetTrueFolderItem("Hard Disk:Photoshop")
g=GetFolderItem("Hard Disk:Photoshop")
EditField1.text=f.Name //the alias to the app
EditField2.text=g.Name //the application
```

See Also

GetFolderItem function, FolderItem class.

GetUniqueIDObjectDescriptor Function

Returns unique AppleEvent Object ID.

Syntax

***result*=GetUniqueIDObjectDescriptor(*DesiredClass*, *Object*, *ID*)**

Part	Type	Description
result	AppleEventObjectSpecifier	AppleEvent object specifier
DesiredClass	String	AppleEvent Class
Object	AppleEventObjectSpecifier	AppleEvent Object
ID	String	ID

See Also

AppleEventObjectSpecifier class.

GOTO Statement

Jumps to a statement label.

Syntax

GOTO *label*

Part	Type	Description
label	String	Label to which to jump.

Note

The **GOTO** statement is included in REALbasic only for compatibility with other implementations of BASIC. Historically, **GOTO** was used extensively in unstructured BASIC to manage flow-of-control. At that time, a program consisted of a single master code segment, with flow-of-control managed by numerous **GOTO** statements that jumped execution around from one section of code to another. In time, such programs proved very hard to maintain because the underlying logic of the program was difficult to follow—especially after the original programmer left the project. The inability to maintain unstructured programs led to the development of event-driven, object-oriented programming environments such as REALbasic.

Example

The following example displays a warning message if Checkbox1 is checked.

```
If checkbox1.value then
 GOTO myCode
End If

//Return used to keep RB from executing the labelled statement anyway
Return
  myCode: //label to jump to
  MsgBox "Unstructured programming is bad"
```

See Also

Exit statement.

Graphics Class

Graphics class objects are used for drawing text, lines, rectangles, ovals, and pictures. Normally you use a graphics object in response to a Paint event, but you can also perform direct drawing by using the **Graphics** property of a Canvas control. Alternatively, you can create vector graphics using the subclasses of the Object2D class.

Super Class

Object

Properties

Name	Type	Description
Bold	Boolean	If True, text will appear in bold style when using the DrawString method.
Copies	Integer	The value in the Copies field of the Print dialog box. This property is only meaningful when the graphics object was created by the OpenPrinterDialog function.
FirstPage	Integer	The value in the From field of the Print dialog box. This property is only meaningful when the graphics object was created by the OpenPrinterDialog function.
ForeColor	Color	The currently selected color for the Graphics object. This color will be used for the various drawing methods.
Height	Integer	The height in pixels of the parent object, typically a Canvas control or a Window. For example, if the graphics class object is in a Canvas control, Height returns the height of the control.
Italic	Boolean	If True, text will appear in italic style when using the DrawString method.
LastPage	Integer	The value in the To field of the Print dialog box. This property is only meaningful when the graphics object was created by the OpenPrinterDialog function.
PenHeight	Integer	The height in pixels used when drawing lines, ovals and rectangles.
PenWidth	Integer	The width in pixels used when drawing lines, ovals and rectangles.
Pixel	X as Integer, Y as Integer	Used to get and set the color of a particular pixel.
TextAscent	Integer	Returns the ascent of a line of text drawn with the current font.
TextFont	String	Used to get and set the current font.
TextHeight	Integer	Contains the height of a line of text drawn with the current font.
TextSize	Integer	Used to get and set the current font size.
Underline	Boolean	If True, text will appear in underline style when using the DrawString method.
UseOldRenderer	Boolean	If True, rendering under Mac OS X is done via QuickDraw rather than Quartz. The property can be modified at runtime. The default for a new object is False. This property may be useful if you need to make your graphics look exactly the same on both the OS 9 and Mac OS X operating systems. The QuickDraw rendering engine is generally faster than Quartz, but Quartz provides the 'native' Mac OS X look.

Name	Type	Description
Width	Integer	The width in pixels of the parent object, typically a Canvas control or a Window. For example, if the graphics class object is in a Canvas control, Width returns the width of the control.

Methods

Name	Parameters	Description
ClearRect	X as Integer, Y as Integer, Width as Integer, Height as Integer	Clears the rectangle described by the parameters passed by filling it with the background color of the parent window.
DrawCautionIcon	X as Integer, Y as Integer	Draws the caution icon at the coordinates specified.
DrawLine	X1 as Integer, Y1 as Integer, X2 as Integer, Y2 as Integer	Draws a line from X1, Y1 to X2, Y2 in the current color.
DrawObject	Object as Object2D, [x as Integer, y as Integer]	Draw the passed Object2D object. The optional parameters X and Y are offsets from the top-left corner of the **Graphics** object.
DrawOval	X as Integer, Y as Integer, Width as Integer, Height as Integer	Draws the outline of an oval in the current color.
DrawPicture	Image as Picture, X as Integer, Y as Integer [,DestWidth as Integer, DestHeight as Integer, SourceX as Integer, SourceY as Integer, SourceWidth as Integer, SourceHeight as Integer]	Draws the picture at the specified location. The optional parameters (surrounded by []) are used to copy a portion of the picture and for scaling the picture. *DestWidth* and *DestHeight* are used to change the scaling of the picture when it's drawn and default to the Width and Height of the picture. *SourceX* and *SourceY* default to 0 and are used to determine the upper-left coordinate you wish to copy from. *SourceWidth* and *SourceHeight* default to the Width and Height of the picture and are used to indicate the portion of the picture you wish to copy.
DrawPolygon	Points() as Integer	Draws a polygon using the values in the 1-based array passed to as the x and y coordinates. Odd numbered array elements are X coordinates and even numbed array elements are Y coordinates.
DrawNoteIcon	X as Integer, Y as Integer	Draws the note icon at the coordinates specified.

Name	Parameters	Description
DrawRect	X as Integer, Y as Integer, Width as Integer, Height as Integer	Draws the outline of a rectangle in the current color. X and Y are the coordinates of the top-left corner.
DrawRoundRect	X as Integer, Y as Integer, Width as Integer, Height as Integer, OvalWidth as Integer, OvalHeight as Integer	Draws the outline of a rounded rectangle in the current color.
DrawStopIcon	X as Integer, Y as Integer	Draws the stop icon at the coordinates specified.
DrawString	Text as String, X as Integer, Y as Integer, [WrapWidth as Integer]	Draws the text at the specified location. The Y parameter specifies the baseline for the text. The optional WrapWidth parameter specifies the width (in pixels) at which text should wrap.
FillOval	X as Integer, Y as Integer, Width as Integer, Height as Integer	Draws an oval filled with the current color.
FillPolygon	Points() as Integer	Draws a polygon using the values in the 1-based array passed as the x and y coordinates. Odd numbered array elements are X coordinates and even numbed array elements are Y coordinates. The polygon is filled with the current ForeColor.
FillRect	X as Integer, Y as Integer, Width as Integer, Height as Integer	Draws a rectangle filled with the current color.
FillRoundRect	X as Integer, Y as Integer, Width as Integer, Height as Integer, OvalWidth as Integer, OvalHeight as Integer	Draws a rounded rectangle filled with the current ForeColor.
NextPage		When printing a graphics object, this method will cause the current page to be printed and a new page to be created.

Name	Parameters	Description
StringHeight	Text as String wrapWidth as Integer	Returns the height of the text (based on the current font and fontsize) and a wrapWidth (pixels). The WrapWidth parameter specifies the width (in pixels) at which text should wrap.
StringWidth	Text as String	Returns the width of the text (based on the current font and font size) in pixels.

Notes

By default, REALbasic uses the Quartz graphics engine on Mac OS X when you call methods of the **Graphics** class such as DrawLine, DrawRect, DrawOval, and so forth. The Quartz graphics engine uses anti-aliasing to make lines look smoother. The disadvantage is that it is slower than line drawing using the older Apple technology. For most applications, the Quartz engine will be fast enough and it does produce more attractive results on Mac OS X. However, if you need more speed, you can improve the performance of the drawing methods under Mac OS X by turning off the Quartz graphics engine and using the QuickDraw engine instead. Do this by setting the UseOldRenderer property of the **Graphics** class to True before you begin drawing.

Examples

This example uses the Paint event handler of a Canvas control to draw the text "The quick brown fox" in Helvetica bold, italics, 18 point, 50 pixels from the top of and 10 pixels from the left side of the control.

```
Sub Paint(g As Graphics)
  g.Bold=True
  g.Italic=True
  g.TextFont="Helvetica"
  g.TextSize=18
  g.DrawString "The quick brown fox", 10, 50
End Sub
```

This example assigns the color of a particular pixel in a Canvas control to a variable.

```
Dim c as Color
c=Canvas1.Graphics.Pixel(10,10)
```

This example sets a particular pixel of a Canvas control to a specific RGB value.

```
Canvas1.Graphics.Pixel(10,10)=RGB(100,105,225)
```

This is example draws a triangle in a Canvas control:

```
Dim Points(6) as Integer
Points(1)=10  //X of Point 1
Points(2)=10  //Y of Point 1
Points(3)=75  //X of Point 2
Points(4)=30  //Y of Point 2
Points(5)=10  //X of Point 3
Points(6)=125 //Y of Point 3
Canvas1.Graphics.ForeColor=RGB(100,200,255)
Canvas1.Graphics.FillPolygon Points
```

The following example assumes that you have imported a resource file into the Project Window that contains a PICT resource whose ID is 129. The following line of code in the Canvas control's Open event handler assigns the picture to the control's Backdrop property:

```
Me.backdrop=app.resourceFork.getpicture(129)
```

Note: Access to the resourcefork is supported only on Macintosh. Check the value of the TargetMacOS constant before attempting to access the resourcefork.

See Also

Canvas control.

Group2D Class

Used to group Object2D objects.

Super Class

Object2D

Properties

Name	Type	Description
Count	Integer	The number of Object2D objects the group contains.
Item	Object2D	Parameter is *Index* as Integer. Returns the Object2D object specified by *Index*.

Methods

Name	Parameters	Description
Append	Object as Object2D	Appends the object passed to it.
Insert	Index as Integer Object as Object2D	Inserts the object passed to it at the specified location.

Name	Parameters	Description
Remove OR	Index as Integer	Removes an object specified by its index.
Remove	Object as Object2D	Removes an object specified by its reference.

Notes

Use the Objects property of the Picture class to associate a **Group2D** object with a picture and the DrawObject method of the Graphics class to draw the object.

A **Group2D** is a container for Object2D shapes (including other **Group2D**s). When a **Group2D** is rotated, translated, or scaled, all of its contents are updated accordingly. This means that all objects use the same coordinate system, but you can quickly transform an entire set of objects by simply transforming their group.

Example

This method adds a PixMapShape and a StringShape to the **Group2D** object and displays it in the window. The picture, "h1", has been dragged to the Project window.

```
Dim px as PixmapShape
Dim s as StringShape
Dim d as Group2D

d=new group2d
px=New PixmapShape(h1)
d.append px

s=New StringShape
s.y=70
s.Text="This is what I call a REAL car!"
s.TextFont="Helvetica"
s.Bold=true
d.append s
graphics.drawobject d,100,100
```

See Also

ArcShape, CurveShape, FigureShape, FolderItem, Graphics, OvalShape, Picture, PixmapShape, RectShape, RoundRectShape, StringShape classes.

Group3D Class

Used to group Object3D objects.

Super Class

Object3D

Methods

Name	Parameters	Description
Count		Number of Object3D objects in the grouped object. Returns an Integer.
Item	Index as Integer	the *Index* item in the grouped object. The first item is index 0. Returns an Object3D.
Append	obj as Object3D	Adds the *obj* object to the grouped object.
Remove	Index as Integer OR obj as Object3D	Removes an object from the group. Removes the *Index* object or the object specified by *obj*. Note that since the Remove method is overloaded, you must always use parentheses around its argument.
Update		Updates the position and orientation of the **Group3D** object.

Notes

The **Group3D** class is used to group 3D objects, but it's also an Object3D object, which allows you to use all the positioning and orienting methods as any other 3D object. When you add an object to a group, it moves and rotates with the group, but this is somewhat expensive, so using groups in this way should be done sparingly. The position and orientation the grouped objects is updated to match the group when you call Update or when the group is drawn in a view. You can remove an object from a group either by its index or by the object reference itself.

See Also

Vector3D, Quaternion, Object3D, Light3D, Bounds3D classes, RB3DSpace control.

GroupBox Control

Draws a radio button group box control. Its primary purpose is to denote a set of related RadioButton controls. Only one RadioButton enclosed by a **GroupBox** can be selected at a time; REALbasic will automatically deselect the other Radio buttons enclosed by the **GroupBox** when one is selected by the user. Radio buttons must be completely enclosed by the **GroupBox** in order to be considered a radio group.

A GroupBox control can also be used to organize other interface elements, but has no effect on their behaviour.

Super Class

RectControl

Because this is a RectControl, see the RectControl for other properties and events that are common to all RectControl objects.

Properties

Name	Type	Description
Bold	Boolean	Applies the bold style to the GroupBox caption.
Caption	String	The GroupBox's text.
Italic	Boolean	Applies the italic style to the GroupBox caption.
TextFont	String	Name of the font used to display the GroupBox caption.
TextSize	**Integer**	Size of the font used to display the GroupBox caption.
Underline	Boolean	Applies the underline style to the GroupBox caption.

Events

Name	Parameters	Description
MouseDown	x as Integer y as Integer	The mouse button was pressed inside the GroupBox at the location passed in to x,y. Returns a Boolean. Return True if you are going to handle the MouseDown. The coordinates x, and y are local to the control. Point 0,0 is the top-left corner of the entire control (to the left of the label and above the box that is drawn).
MouseDrag	x as Integer y as Integer	The mouse button was pressed inside the GroupBox and moved (dragged) at the location local to the control passed in to x,y. This event will not occur unless you return True in the MouseDown event.
MouseUp	x as Integer y as Integer	The mouse button was released inside the GroupBox at the location passed in to x,y. This event will not occur unless you return True in the MouseDown event.

Notes

If the Caption property contains an ampersand character, it will display only if it is preceded by another ampersand character. This is done to make Macintosh and Windows applications behave consistently.

Some versions of Mac OS (8.0, 8.1, and OS X Public Beta) have a bug which causes a GroupBox to obscure any controls behind it. To avoid this bug, use the Format ▶ Move To Back command to move each GroupBox control behind the controls they contain.

Example

See the RadioButton control.

See Also

RadioButton control.

GuessJapaneseEncoding Function

Guesses the text encoding used for a sample of Japanese text (2022_JP, EUC, S-JIS, or plain US_ASCII). If it is given a string whose encoding is already known to REALbasic, it returns that encoding. If the encoding is unknown, it assumes the encoding is some Japanese script, and attempts to guess which one.

Syntax

***result*=GuessJapaneseEncoding(*text*)**

Part	Type	Description
result	TextEncoding	The guessed TextEncoding.
text	String	The text whose text encoding will be guessed.

Hex Function

Returns the hexadecimal version of the number passed as a string.

Syntax

***result*=Hex(*value*)**

Part	Type	Description
result	String	*Value* converted to hexadecimal.
value	Integer	The number to be converted to hexadecimal.

Notes

If the value is not a whole number, the decimal value will be truncated.

You can specify binary, hex, or octal numbers by preceding the number with the & operator and the letter that indicates the number base. The letter b indicates binary, h indicates hex, and o indicates octal.

VB Compatibility Note: VB rounds the value to the nearest whole number so the **Hex** function will probably be changed in a future release to do this as well.

Examples

Below are examples of various numbers converted to hex:

```
Dim hexVersion As String
hexVersion=Hex(5) //returns "5"
hexVersion=Hex(75) //returns "4B"
hexVersion=Hex(256) //returns "100"
```

See Also

& operator; Bin and Oct functions.

HighlightColor Function

Returns the color the user has chosen as their highlight color.

Syntax

***result*=HighlightColor**

Part	Type	Description
result	Color	The user's chosen highlight color.

Notes

If the user is running Mac OS 8 –9.2, the highlight color is set using the Appearance Control Panel. In Mac OS X, the highlight color is set in the General System Preferences panel.

See Also

DarkBevelColor, DarkTingeColor, FillColor, FrameColor, LightBevelColor, TextColor functions.; Color data type.

HSV Function

Returns a Color object based on the HSV (hue, saturation, value) color model.

Syntax

***result*=HSV(*hue*, *saturation*, *value*)**

Part	Type	Description
result	Color	An object that represents the color based on the *hue*, *saturation*, and *value* values.
hue	Double	The value of Hue in the color.
saturation	Double	The value of Saturation in the color.
value	Double	The value of Value in the color.

Notes

The **HSV** function returns a color object based on the amounts of Hue, Saturation, and Value passed. These amounts are represented by doubles between 0 and 1.

Colors can also be created using the RGB or CMY models.

Examples

This example uses the **HSV** function to assign a Color to the FillColor property of a Rectangle control.

```
Rectangle1.FillColor=HSV(.8,.5,.75)
```

See Also

Color data type, RGB, CMY, SelectColor functions.

HTTPSocket Class

Used to send and receive data via the HTTP protocol.

Super Class

TCPSocket

Properties

None

Methods

Name	Parameters	Description
ClearRequestHeaders		Clears all set request headers. See the description of the SetRequestHeaders method for information on headers.
EncodeFormData	Form as Dictionary	Returns a String. Takes the key/value pairs from the specified Dictionary and formats them into a URL-encoded String. The result can then be used in the URL to send data to a form with a GET action or as content for a POST.
Get	URL as String [File as FolderItem]	Requests the content from the specified URL. The PageReceived event will execute when the data has been retrieved. If a FolderItem is passed, the data will be downloaded to that file. The DownloadComplete event will execute when the response data has been retrieved.
Post	URL as String [File as FolderItem]	Issues a POST command to the web server. The PageReceived event will execute when the response data has been retrieved. If a FolderItem is passed, the data will be downloaded to that file. The DownloadComplete event will execute when the response data has been retrieved.
SetFormData	Form as Dictionary	Takes the key/value pairs from the Dictionary and converts it to a URL-encoded String. This method also sets the action to POST. After using this method, the form can be sent to the server with the POST action.
SetPostContent	Content as String, ContentType as String	Sets the content data and content type to be sent to the server for a POST command.

Name	Parameters	Description
SetRequestHeader	Name as String, Value as String	Sets the value of a request header. Request headers are sent to the server along with the action (GET, POST, etc.). They can contain data that the server might use to process your request. Some examples of common request headers are for things such as referring pages, the client name, OS CPU, and so forth.

Events

Name	Parameters	Description
Connected		Executes when the connection is established with the web server.
DownloadComplete	URL as String, HTTPStatus as Integer, Headers as Dictionary, File as FolderItem	Executes when a download is complete after using the Get method. URL contains the URL that was downloaded. *File* is the FolderItem the data was downloaded to, and *Headers* is a dictionary of the HTTP headers that were returned by the server. The HTTPStatus code is also passed to this event. These codes are used for messages such as "Page not found" (error 404), and so forth.
Error	Code as Integer	Executes when an error occurs. All the standard TCPSocket error codes are passed in *Code*. An additional error code is: 1- Download file could not be created.
HeadersReceived	Headers as Dictionary	Executes when the headers have been retrieved from the server.
PageReceived	URL as String, HTTPStatus as Integer, Headers as Dictionary, Content as String	Executes when a new page has been retrieved from the server. It provides the URL that was requested and the HTTP headers from the web server. The HTTPStatus code is also passed to this event. These codes are used for messages such as "Page not found" (error 404), and so forth.

Name	Parameters	Description
ReceiveProgress	BytesReceived as Integer, TotalBytes as Integer	Executes periodically during the download process. It provides the current number of bytes received and the total number of bytes (if available).
SendProgress	BytesSent as Integer, BytesLeft as Integer	Executes periodically during the upload process. It provides the number of bytes sent and the number of bytes left to send.

Notes

Use the **HTTPSocket** class to send and receive data via the HTTP protocol. Since this is a subclass of the TCPSocket class, all the standard socket APIs are also available. To perform secure communications via HTTP, use the HTTPSecureSocket class instead. It is otherwise identical, except that it is subclassed from SSLSocket instead of TCPSocket. This makes all the options for encrypted communications available to HTTPSecureSocket objects.

Example

The following example retrieves the specified URL:

```
Dim socket1 as HTTPSocket
socket1 =New HTTPSocket
socket1.get "http://www.realsoftware.com/"
```

The following example posts a simple form:

```
Dim form as Dictionary
Dim socket1 as HTTPSocket
socket1=New HTTPSocket

  // create and populate the form object
  form = New Dictionary
  form.value("firstname") = "Bob"
  form.value("lastname") = "Brown"

  // setup the socket to POST the form
  socket1.setFormData form
  socket1.post "http://www.myformlocation.com/form.php"
```

See Also

SocketCore, TCPSocket, HTTPSecureSocket classes.

HTTPSecureSocket Class

Used to do secure transmissions via the HTTP protocol using the SSL or TLS protocols.

Super Class

SSLSocket

Properties

None

Methods

Name	Parameters	Description
ClearRequestHeaders		Clears all set request headers. See the description of the SetRequestHeaders method for information on headers.
EncodeFormData	Form as Dictionary	Returns a String. Takes the key/value pairs from the specified Dictionary and formats them into a URL-encoded String. The result can then be used in the URL to send data to a form with a GET action or as content for a POST.
Get	URL as String [File as FolderItem]	Requests the content from the specified URL. The PageReceived event will execute when the data has been retrieved. If a FolderItem is passed, the data will be downloaded to that file. The DownloadComplete event will execute when the response data has been retrieved.
Post	URL as String [File as FolderItem]	Issues a POST command to the web server. The PageReceived event will execute when the response data has been retrieved. If a FolderItem is passed, the data will be downloaded to that file. The DownloadComplete event will execute when the response data has been retrieved.
SetFormData	Form as Dictionary	Takes the key/value pairs from the Dictionary and converts it to a URL-encoded String. This method also sets the action to POST. After using this method, the form can be sent to the server with the POST action.
SetPostContent	Content as String, ContentType as String	Sets the content data and content type to be sent to the server for a POST command.

Name	Parameters	Description
SetRequestHeader	Name as String, Value as String	Sets the value of a request header. Request headers are sent to the server along with the action (GET, POST, etc.). They can contain data that the server might use to process your request. Some examples of common request headers are for things such as referring pages, the client name, OS CPU, and so forth.

Events

Name	Parameters	Description
Connected		Executes when the connection is established with the web server.
DownloadComplete	URL as String, HTTPStatus as Integer, Headers as Dictionary, File as FolderItem	Executes when a download is complete after using the Get method. URL contains the URL that was downloaded. File is the FolderItem the data was downloaded to, and Headers is a dictionary of the HTTP headers that were returned by the server. The HTTPStatus code is also passed to this event. These codes are used for messages such as "Page not found" (error 404), and so forth.
Error	Code as Integer	Executes when an error occurs. All the standard TCPSocket error codes are passed in Code. An additional error code is: 1- Download file could not be created.
PageReceived	URL as String, HTTPStatus as Integer, Headers as Dictionary, Content as String	Executes when a new page has been retrieved from the server. It provides the URL that was requested and the HTTP headers from the web server. The HTTPStatus code is also passed to this event. These codes are used for messages such as "Page not found" (error 404), and so forth.
ReceiveProgress	BytesReceived as Integer, TotalBytes as Integer	Executes periodically during the download process. It provides the current number of bytes received and the total number of bytes (if available).

Notes

Use the **HTTPSecureSocket** class to perform secure communications via the HTTP protocol. It is identical to the HTTPSocket class except that it is derived from the SSLSocket class instead of the TCPSocket class. This means that you can use the Secure property of SSLSocket to switch to HTTPS. When you set Secure to True, the default port changes from 80 to 443.

Example

The following example posts a simple form:

```
Dim form as Dictionary
Dim socket1 as HTTPSecureSocket
Socket1.Secure=True

  // create and populate the form object
  form = New Dictionary
  form.value("firstname") = "Bob"
  form.value("lastname") = "Brown"

  // setup the socket to POST the form
  socket1.setFormData form
  socket1.post "https://www.myformlocation.com/form.php"
```

See Also

SocketCore, TCPSocket, SSLSocket, HTTPSocket classes.

If...Then...Else Statement

Conditionally executes a group of statements, depending on the value of a boolean expression.

Syntax

If *condition* Then

statements

[ElseIf *condition-n* Then

***elseIfStatements*]...**

[Else

***elseStatements*]**

End [If]

OR

If *condition* Then *statement* [Else] [*statement]*

Part	Description
condition	Required. A boolean, numeric, or string expression that evaluates to True or False.
statements	Optional. One or more statements that are executed if *condition* is True.
condition-n	Optional. Same as *condition*.
elseIfStatements	Optional. One or more statements executed that are executed if the associated *condition-n* is True.
elseStatements	Optional. One or more statements executed if no previous *condition* or *condition-n* expression is True.

Notes

When executing an **If** statement, the condition is tested. **If** condition is True, the statements associated with the **If** statement following the **Then** statement are executed. **If** condition is False and an **Else** clause follows, its statements will be executed. **If** condition is False and there is no **Else** clause or it is preceded by an **ElseIf** statement, the condition following the **ElseIf** statement is tested. After executing the statements following **Then**, **ElseIf** or **Else** execution continues with the statement that follows **End If**.

An If statement can be written on one line, provided the code that follows the **Then** and **Else** statements can be written on one line. Using this syntax, you omit the **End if** statement. For example, the following examples are valid:

```
If error=123 Then MsgBox "An error occured."
If error=123 Then MsgBox "An error occured." Else MsgBox "Never mind!"
```

The following is not valid because the Then clause requires two lines:

```
If error=123 Then Beep MsgBox "An error occured."
```

Use the syntax shown in the first example.

Examples

This example shows an **If** statement.

```
If error=-123 Then
  Beep
  MsgBox "Whoops! An error occured."
End If
```

This example shows an **If** statement that includes the use of **ElseIf** and **Else** clauses.

```
Dim theNumber As Integer
Dim digits As Integer
theNumber=33
If theNumber<10 Then
  digits=1
ElseIf theNumber<100 Then
  digits=2
Else
  digits=3
End If
```

See Also

Select Case statement.

IllegalCastException Runtime Error

Caused by an attempt to recast an object and then sending it a message that its original type cannot handle.

Super Class

RuntimeException class

Notes

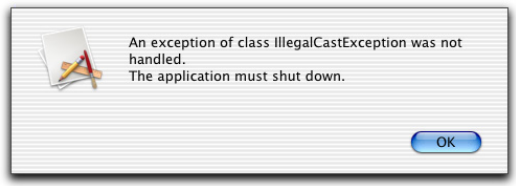
An IllegalCastException error occurs when you attempt to cast an object to a different type and call a method or access a property belonging only to the original type. For example, if you recast a BevelButton as a PushButton (see the example below) and then call the Pushbutton's Push method, an IllegalCastException will occur because a BevelButton isn't a PushButton and happens not to have a Push method.

Example

This example attempts to cast a BevelButton as a PushButton then call the Push method of the PushButton class. This example presumes that there is a BevelButton called "BevelButton1" in the same window where this code is being called:

```
Dim c as Control
c = BevelButton1
PushButton(c).Push
```

When you test this code in the IDE, execution will stop at the last line and display an IllegalCastException error in your Code editor. If you test the code in a standalone application, the application will display a generic error message and quit when the user clicks OK.



You can handle the error by adding an Exception block to your code:

```
Dim c as Control
c = BevelButton1
PushButton(c).Push
Exception err
If err isA IllegalCastException then
 MsgBox "Trying to recast a BevelButton as PushButton!"
end if
```

When you test this code in a standalone application, the message box displays a more informative message.

The key difference is that, when the user accepts this dialog box, the application does not quit.

Moral of the story: try to handle exception errors.

See Also

RuntimeException class, OutofBoundsException error, NilObjectException error, StackOverFlowException error, TypeMismatchException error, Function statement, Raise statement, Nil function, Exception block.

ImageWell Control 

Adds an ImageWell to a window, which can display a PICT image on Macintosh or BMP image on Windows. If QuickTime is installed on the user's machine, the additional picture formats supported by QuickTime can also be used.

Super Class

RectControl

Because this is a RectControl, see the RectControl for other properties and events that are common to all RectControl objects.

Properties

Name	Type	Description
Image	Picture	Picture displayed in the ImageWell. On Macintosh, *Picture* must be a PICT.

Example

The following line of code in the ImageWell's Open event displays a picture of a prancing stallion in the ImageWell. The image had been dragged into the Project window.

```
me.image=Ferrari
```

The following example implements drag and drop between two ImageWells, a PICT file dragged from the desktop to either ImageWell, and either ImageWell's image dragged to the desktop to create a clipping.

In the Open event handler of the ImageWells, the two statements tell the controls to accept either a dragged picture or a dragged file of type PICT (the file type "image/x-pict") was defined previously in the File Types dialog box).

```
Me.AcceptPictureDrop
Me.AcceptFileDrop("image/x-pict")
```

The DropObject event handler is:

```
Sub Obj as DragItem
 If Obj.PictureAvailable then
  Me.Image=Obj.Picture
 elseif Obj.FolderItemAvailable then
  Me.Image=Obj.FolderItem.OpenAsPicture
 End if
End Sub
```

The MouseDown event handler is:

```
Dim d as DragItem
d=NewDragItem(Me.left,Me.top,Me.width,Me.height)
d.picture=Me.Image
d.Drag //Allow the drag
```

See Also

RectControl class.

Insert Method

Inserts a new element into an array.

Syntax

array*.Insert *index*, *value

Part	Type	Description
array	array of any type	Required. The array in which to insert the new element.
index	Integer	The position in *array* to insert the new element.
value	Same as array type	The value to be assigned to the new *array* element.

Notes

All items below and including *index* are moved down one position to make room for the new element.

The **Insert** method works with one-dimensional arrays only.

Examples

This example inserts a new element at position 5 into the array aNames and assigns it the string "Bill".

```
aNames.Insert 5, "Bill"
```

See Also

Append method, Dim statement, Redim statement, Remove method, UBound function, Sort method.

InStr Function

Returns the position of the first occurrence of a string inside another string. The first character is numbered 1.

Syntax

***result*=InStr([*start*,] *source*, *find*)**

OR

***result*=*stringVariable*.InStr([*start*,] *find*)**

Part	Type	Description
result	Integer	The position of the first occurrence of *find* in *source*.
start	Integer	Optional position from which to begin searching the source string. One is the default if omitted.
source	String	Required. String expression being searched.
find	String	Required. String expression being sought.
stringVariable	String	Any variable of type String.

Notes

If the *find* string is not found within the *source* string, 0 (zero) is returned. If the *find* string is null, then *start* is returned. That is, **InStr**("This","") returns 1 and **InStr**(3,"This","") returns 3.

InStr is case-insensitive.

If you need to find the byte position of the *find* string within the *source* string, use the InStrB function or you need a case-sensitive function.

Examples

This example uses the **InStr** function to locate a string within another string.

```
Dim first As Integer
first = InStr("This is a test", "t")
//returns 1
first = InStr("This is a test", "is")
//returns 3
first = InStr(4, "This is a test", "is")
//returns 6
first = InStr("This is a test", "tester")
//returns 0
```

See Also

Asc, Chr, Left, Len, Mid, Right, StrComp functions.

InStrB Function

Returns the byte position of the first occurrence of a string inside another string. The first character is numbered 1.

Syntax

***result*=InStrB([*start*,] *source*, *find*)**

OR

***result*=*stringVariable*.InStrB([*start*], *find*)**

Part	Type	Description
result	Integer	The byte position of the first occurrence of *find* in *source*.
start	Integer	Optional byte position from which to begin searching the source string. One is the default if omitted.
source	String	Required. String expression being searched.
find	String	Required. String expression being sought.
stringVariable	String	Any variable of type String.

Notes

If the *find* string is not found within the *source* string, 0 (zero) is returned. **InStrB** is case-sensitive; it treats *source* as a series of raw bytes. It should be used instead of InStr when the string represents binary data or when your application will run in a one-byte character set (such as the US system) and you want case-sensitivity.

If you need to find the character position of the *find* string within the *source* string, use the InStr function.

Examples This example uses the **InStrB** function to locate a string within another string.

```
Dim first As Integer
first = InStrB("This is a test", "T")
//returns 1
first = InStrB("This is a test", "t")
//returns 11
first = InStrB("This is a test", "is")
//returns 3
first = InStrB(4, "This is a test", "is")
//returns 6
first = InStrB("This is a test", "tester")
//returns 0
first = InStrB("This Is a test", "Is")
//returns 6
```

See Also AscB, ChrB, InStr, LeftB, LenB, MidB, RightB, StrComp functions.

Integer Data Type

An **Integer** is an intrinsic data type in REALbasic. An **Integer** is a whole number between ±2,147,483,648. In other programming languages, REALbasic's Integer type is a long integer. Because integers are numbers, you can perform mathematical calculations on them. An **Integer** value uses 4 bytes of memory.

See Also Single, Double, String, Boolean, Vartype data types; -, +, *, /, <, <=, =, >=, >, <>, \, Mod, Str, Val functions.

InvalidParentException Error

Super Class RuntimeException

Notes You tried to get the parent of a control using the Parent property of the Control class, but its parent is in a different window. The parent control must be in the same window as the control or, if not control is the parent, the parent is the containing window.

See Also Parent property of the Control class; Exception block; RuntimeException class.

Is Operator

Compares two object references to determine whether they both refer to the same object.

Syntax

result=object1* Is *object2

Part	Type	Description
result	Boolean	Any container expecting a boolean value.
object1	Object	Any object name.
object2	Object	Another object name.

Notes

The **Is** operator returns True if object1 and object2 actually refer to the same object. It checks identity, not contents, so it is not affected by the presence of a comparison operator. Use it the same way as the "=" operator.

Example

In the following example, the **Is** operator returns True.

```
Dim w1,w2 as Window
w1=Window1
w2=Window1
If w1 Is w2 then
//do something here
Else
//do something else here
End if
```

See Also

IsA operator.

IsA Operator

Used to determine the class of a particular object reference.

Syntax

result=object* IsA *object class

Part	Type	Description
result	Boolean	Any container expecting a boolean value.
object	Object	Any object name.
object class	Object class	Any object class.

Notes

If *object* is of type *object class*, the **IsA** operator returns True; if not, it returns False.

The **IsA** operator returns True for the object's own class as well as the super class from which that class was derived, all the way to the Object class. **IsA** will report that all classes ultimately derive from Object.

For example, suppose you create a subclass of EditField called SecureEditField, that disables the Cut and Copy commands in the Edit menu. If EditField1 is an instance of SecureEditField, the tests:

```
EditField1 IsA EditField
```

and

```
EditField1 IsA SecureEditField
```

both return True.

You use the IsA operator to *cast* objects. With the **IsA** operator, you test whether an object is of a specific subclass and, if it is, cast it as that type to do something specific with it.

Here is an example that uses a For loop to cycle through all the controls in a window to test whether each control is an EditField. If it is, it casts the control, gets its name and the value of its Text property, and assigns the contents of the EditField to the field in a database table named *fieldname*.

```
Dim r as DatabaseRecord
Dim fieldname, fieldContents as String
.
.
For i = 1 to Self.ControlCount  //number of controls in window
  If Self.control(i) isa EditField then
    fieldname = EditField(control(i)).DataField //cast it
  //the text property assigned to the contents of that field
    fieldContents = EditField(control(i)).text
  r.column(fieldname)= fieldContents
  end if
next
```

As you can see, the code is generic since it uses no references to specific EditFields or databases.

A second example of casting uses the Window function to get the frontmost window and then using **IsA** to determine which what kind of window it is (i.e., Window1, Window2) and then casts it as that type to access something specific about the window

```
If Window(0) IsA Window1 then
  Window1(Window(0)).Pushbutton1.enabled = True
End if
```

Example

This example uses the **IsA** operator to determine if the variable p is a PushButton object. If it is, the caption property is assigned "OK".

```
If p IsA PushButton Then
  p.caption="OK"
End if
```

See Also

Please refer to "Creating Reusable Objects with Classes" on page 317 of the Developer's Guide.

IsCMMClick Function

Returns True if the MouseDown event occured while the Control key was pressed (indicating that the user wishes to display any contextual menu that might exist for the object being clicked on) or, on Windows, whether the right mouse button was clicked. It works only for windows and controls that have a MouseDown event handler.

Syntax

***result*=IsCMMClick**

Part	Type	Description
result	Boolean	True if the mouse was clicked while the Control key was down and False if it was not. On Windows, the function returns True if the right mouse button is clicked in a window or control that has a MouseDown event handler.

Notes

This function is used to determine if the user wishes to view any available contextual menus for the object they are clicking on. For more information on implementing contextual menus, see the ContextualMenu Control.

Examples

This example checks to see if the user has pressed the Control key, then displays a contextual menu.

```
If IsCMMClick Then
  ContextualMenu1.Open //display the contextual menu
End If
```

See Also ContextualMenu control.

Keyboard Object

An intrinsic object that represents the current state of the keyboard.

Properties

Name	Type	Description
AsyncCommandKey	Boolean	If True, the Command key is depressed.
AsyncControlKey	Boolean	If True, the Control key is depressed.
AsyncKeyDown	Boolean	If True, the key whose KeyCode was passed, is depressed.
AsyncOptionKey	Boolean	If True, the Option key is depressed.
AsyncShiftKey	Boolean	If True, the Shift key is depressed.
CommandKey	Boolean	If True, the Command key was depressed when the current method or event handler began.
ControlKey	Boolean	If True, the Control key was depressed when the current method or event handler began.
OptionKey	Boolean	If True, the Option key was depressed when the current method or event handler began.
ShiftKey	Boolean	If True, the Shift key was depressed when the current method or event handler began.

Events

None

Methods

None

Notes

The **Keyboard** object is used to determine if particular keys are being pressed. The CommandKey, ControlKey, OptionKey, and ShiftKey properties should be used when you only need to know if one of these modifier keys was pressed when the current method or event handler was executed. If you are going to test for any of these keys (or for any other key on the keyboard) more than once (like in a loop), use the AsyncCommandKey, AsyncControlKey, AsyncKeyDown, AsyncOptionKey, and AsyncShiftKey properties. These properties are updated constantly during code execution.

Windows

You can pass raw Win32 key codes to AsyncKeyDown. Add 1 to the raw keyCode and put it in the high word of the Integer you are passing. For example:

```
Keyboard.AsyncKeyDown((rawKeyCode + 1) * &hFFFF)
```

The 'regular' way of detecting a particular keyCode, illustrated by the first example in the Examples section, also works for Windows.

The following illustration shows the keycodes used with the **Keyboard** object. The AsyncKeyDown property returns True when the keycode passed to it is pressed.

Examples

The following example tests whether the key for the letter "A" was pressed:

```
If Keyboard.AsynckeyDown(&h00) then
 //do somthing with this key here
end if
```

This example displays the command key shortcut for a button when the Command key is held down. The code is in the Action event handler of a Timer control that checks the CommandKey property twice a second:

```
If Keyboard.CommandKey Then
  PushButton1.Caption="Save "+Chr(17)+"S"
Else
  PushButton1.Caption="Save"
End if
```

This example monitors the arrow keys and detects when an arrow key is pressed. Place it in the EnableMenuItems event of the Application class and it will work even if no windows are open.

```
If Keyboard.AsyncKeyDown(123) then
   //do something here...
end if
If Keyboard.AsyncKeyDown(124) then
  //do something here...
end if
If Keyboard.AsyncKeyDown(125) then
   //do something here...
end if
If Keyboard.AsyncKeyDown(126) then
  //do something here...
end if
```

KeyChain Class

Gives you access to the default Mac OS "classic" and Mac OS X Keychains for your applications. The **KeyChain** class does not provide access to internet passwords.

Super Class

Object

Constructor

Name	Parameters	Description
KeyChain	Index as Integer	Gives you a reference to any **KeyChain** known by the KeyChain manager. The index value should be in the range from 0 to System.KeyChainCount.

Properties

Name	Type	Description
Handle	Integer	Returns a Handle to the **KeyChain**. Useful only if you want to refer to REALbasic's Keychains with Declare statements.

Methods

Name	Parameters	Description
AddPassword	kci as KeyChainItem Password as String	Adds the password to the **Keychain** and associates it with a KeyChainItem. If it fails, it generates a KeyChainException.

Name	Parameters	Description
FindPassword	ByRef kci as KeyChainItem	Returns a String. Attempts to find the password for an Item in the given Keychain that matches *kci*. At the end of a successful call, *kci* will point to the found KeyChain Item. If it fails, it generates a KeyChainException.
Lock	LockAll as Boolean	Locks the Keychain. If LockAll is True, it will lock all the Keychains on the system. Note from Apple: There is usually no need for an application to ever call Lock. Unless your application is directly responding to a user's request for a keychain to be locked, it is recommended that you leave the keychain unlocked so that the user does not have to unlock it again in another application. If it fails, it generates a KeyChainException.
Unlock	Password as String	Unlocks the keychain if the password is the empty string (" "). The system displays the standard Unlock dialog, and the passed keychain will appear as the selected item in that dialog's pop-up menu. NOTE: because of the way Apple has implemented this, if you pass an incorrect password once to a particular keychain, the system will require human interaction from thenceforth on that keychain until the system is rebooted. If it fails, it generates a KeyChainException.

Notes

The *keychain* is a system wide facility on Mac OS 8.5 and above (including Mac OS X) to store account passwords for applications. By taking advantage of the built-in keychain facility, your users won't have to type their password if their keychain is unlocked. You should always ask the user before storing something in the keychain.

An equivalent technology to the Mac OS KeyChain doesn't currently exist on the Windows platform, so the KeyChain class is not supported on Windows.

Examples

The following example adds a KeyChainItem for an application and assigns a password:

```
Dim NewItem as KeyChainItem
  If System.KeyChainCount > 0 then

    NewItem = New KeyChainItem
    'Indicate the name of the application
    NewItem.ServiceName = "MyApplication"

    'Create a new keychain item for the application and assign the password
    System.KeyChain.AddPassword NewItem, "SecretPassword"
  Else
    Beep
    MsgBox "You don't have a key chain."
  End if

Exception err as KeyChainException
  MsgBox "Can't add item: " + err.Message
```

The following example retrieves the password and displays it in a message box.

```
Dim ItemToFind as KeyChainItem
Dim password As String

ItemToFind = New KeyChainItem

'Indicate the name of the application whose keychain item you wish to find
ItemToFind.ServiceName = "MyApplication"

  'get application's password from the system keychain
password = System.KeyChain.FindPassword(ItemToFind)
MsgBox "The password for this item is: " + password

Exception err as KeyChainException
MsgBox "Can't find item: " + err.Message
```

See Also

KeyChainItem class; KeyChainException error; System object.

KeyChainException Error

A method of the KeyChain or KeyChainItem classes failed. See the error code returned by **KeyChainException** to diagnose the problem.

Super Class

RuntimeException

Properties

Name	Type	Description
ErrorNumber	Integer	Indicates the type of error that was encountered.

Notes

The following tables shows the values of *ErrorNumber* and the associated text returned in the Message property.

Error Number	Message
-128	User Cancelled.
-25291	Key Chain not available.
-25292	Key Chain read only.
-25293	Key Chain Authorization failed.
-25294	No such Key Chain.
-25295	Invalid Key Chain.
-25296	Duplicate Key Chain.
-25797	Key Chain Duplicate Callback.
-25298	Key Chain Invalid Callback.
-25299	Key Chain Duplicate Item.
-25300	Key Chain Item Not Found.
-25301	Key Chain Buffer Too Small.
-25302	Key Chain Data Too Large.
-25303	Key Chain No Such Attribute.
-25304	Key Chain Invalid Item Reference.
-25305	Key Chain Invalid Search Reference.
-25306	Key Chain No Such Class.
-25307	No Default Key Chain.
-25308	Key Chain Interaction Not Allowed.
-25309	Key Chain Read Only Attribute.
-25310	Wrong Key Chain Version.
-25311	Key Chain Key Size Not Allowed.
-25312	Key Chain No Storage Module.

Error Number	Message
-25313	Key Chain No Certificate Module.
-25314	Key Chain No Policy Module.
-25315	Key Chain Interaction Required.
-25316	Key Chain Data Not Available.
-25317	Key Chain Data Not Modifiable.
-25318	Key Chain Create Chain Failed.

Examples

The following example displays a message box if, for example, you try to create more than one KeyChainItem for the same application:

```
Dim NewItem as KeyChainItem
  If System.KeyChainCount > 0 then

    NewItem = New KeyChainItem
    'Indicate the name of the application
    NewItem.ServiceName = "MyApplication"

    'Create a new keychain item for the application and assign the password
    System.KeyChain.AddPassword NewItem, "SecretPassword"
  Else
    Beep
    MsgBox "You don't have a key chain."
  End if

Exception err as KeyChainException
  MsgBox err.Message+". Error Code: "+Str(err.errorNumber)
```

The following example uses an Exception block to display a message box if the application specified by ServiceName does not have a KeyChainItem.

```
Dim ItemToFind as KeyChainItem
Dim password As String

ItemToFind = New KeyChainItem

'Indicate the name of the application whose keychain item you wish to find
ItemToFind.ServiceName = "MyApplication"

  'get application's password from the system keychain
password = System.KeyChain.FindPassword(ItemToFind)
MsgBox "The password for this item is: " + password

Exception err as KeyChainException
MsgBox "Can't find item: " + err.Message
```

See Also KeyChain, KeyChainItem, RuntimeException classes; System object.

KeyChainItem Class

Refers to a Keychain item.

Super Class

Object

Properties

Name	Type	Description
AccountName	String	Contains the name of the account (required for adding, can be Nil to find).
Comment	String	End user editable string containing comments for this Keychain item.
Description	String	End user visible string describing this Keychain item.
Handle	Integer	Contains the **KeyChainItem** reference, for use with Toolbox calls.
Label	String	End user editable string containing the label for this Keychain item.
ServiceName	String	Contains the name of the service (required for adding, can be Nil to find). To add a password for an application, set ServiceName to the application's name.

Methods

Name	Parameters	Description
Delete		Deletes the **KeyChainItem**. You can change passwords in a KeyChain by deleting the original item using this method and adding another item.

Notes

REALbasic KeyChainItems can access passwords for applications only, not internet passwords.

Examples

The following example adds a KeyChainItem for an application and assigns a password:

```
Dim NewItem as KeyChainItem
  If System.KeyChainCount > 0 then

    NewItem = New KeyChainItem
    'Indicate the name of the application
    NewItem.ServiceName = "MyApplication"

    'Create a new keychain item for the application and assign the password
    System.KeyChain.AddPassword NewItem, "SecretPassword"
  Else
    Beep
    MsgBox "You don't have a key chain."
  End if

Exception err as KeyChainException
  MsgBox "Can't add item: " + err.Message
```

The following example retrieves the password that was set in the previous example and displays it in a message box.

```
Dim ItemToFind as KeyChainItem
Dim password As String

ItemToFind = New KeyChainItem

'Indicate the name of the application whose keychain item you wish to find
ItemToFind.ServiceName = "MyApplication"

  'get application's password from the system keychain
password = System.KeyChain.FindPassword(ItemToFind)
MsgBox "The password for this item is: " + password

Exception err as KeyChainException
MsgBox "Can't find item: " + err.Message
```

See Also

KeyChain class, KeyChainException error; System object.

KeyNotFoundException Error

You tried to access an item in a Dictionary using a key that is not in the Dictionary.

Super Class

RuntimeException.

Example

The following For loop produces a **KeyNotFoundException** error when the loop runs with the counter equal to 2. The value of 2 — which is a Variant, not an Integer — has just been removed.

```
Dim d as Dictionary
Dim i as Integer
d=New Dictionary
d.Value(1)="ID"
d.Value(2)="Lois Lane"
d.Value(3)="Reporter"
d.Value(4)=84000
d.Remove(2)
For i=1 to d.Count //fails when i=2
 ListBox1.Addrow d.value(i)
Next
```

To handle the error, add an Exception block to the end of the method, such as:

```
Exception err as KeyNotFoundException
 MsgBox "You tried to access a nonexistent item!"
```

See Also

Dictionary, RuntimeException classes; Exception block.

Left Function

Returns the first *n* characters in a source string.

Syntax

***result*=Left(*source*, *count*)**

OR

***result*=*stringVariable*.Left(*count*)**

Part	Type	Description
result	String	The first *count* characters of *source*.
source	String	The source string from which to get the characters.

Part	Type	Description
count	Integer	The number of characters you wish to get from *source*. If *count* is greater than the number of characters in *source*, all characters are returned.
stringVariable	String	Any variable of type String.

Notes

The **Left** function returns characters from the *source* string starting from the left side (as the name implies).

If you need to read bytes rather than characters, use the LeftB function.

Example

This example returns the first five characters in a string.

```
Dim s As String
s=Left("Hello World", 5) //returns "Hello"
```

See Also

Asc, Chr, InStr, Len, LeftB, Mid, Right functions.

LeftB Function

Returns the first *n* bytes in a source string.

Syntax

***result*=LeftB(*source*, *count*)**

OR

***result*=*stringVariable*.LeftB(*count*)**

Part	Type	Description
result	String	The first *count* bytes of *source*.
source	String	The source string from which to get the bytes.
count	Integer	The number of bytes you wish to get from *source*. If *count* is greater than the number of bytes in source, all bytes are returned.
stringVariable	String	Any variable of type String.

Notes

The **LeftB** function returns bytes from the *source* string starting from the left side (as the name implies).

If you need to read characters rather than bytes, use the Left function.

Example

This example uses the **LeftB** function to return the first 5 bytes from a string.

```
Dim s As String
s=LeftB("Hello World", 5) //returns "Hello"
```

See Also

AscB, ChrB, InStrB, Left, LenB, MidB, RightB functions.

Len Function

Returns the number of characters in the specified string.

Syntax

***result*=Len(*string*)**

OR

***result*=*stringVariable*.Len**

Part	Type	Description
result	Integer	The number of characters in *string*.
string	String	Any valid string expression.
stringVariable	String	Any variable of type String.

Notes

If you need the number of bytes in the string rather than the number of characters, use the LenB function.

Examples

This example uses the **Len** function to return the number of characters in a string.

```
Dim n As Integer
n=Len("Hello world") //returns 11
```

See Also

Asc, Chr, InStr, LenB, Mid, Right functions.

LenB Function

Returns the number of bytes in the specified string.

Syntax

***result*=LenB(*string*)**

OR

***result=stringVariable*.LenB**

Part	Type	Description
result	Integer	The number of bytes in *string*.
string	String	Any valid string expression.
stringVariable	String	Any variable of type String.

Notes

LenB treats *string* as a series of bytes, rather than a series of characters. It should be used when *string* represents binary data. If you need to know the number of characters in *string* rather than the number of bytes, use the Len function.

Examples

This example uses the **LenB** function to return the number of bytes in a string.

```
Dim n As Integer
n=LenB("Hello world") //returns 11
```

See Also

AscB, ChrB, InStrB, Len, MidB, RightB functions.

Light3D Class

Allows you to define your own lights for use in RB3DSpace worlds.

Super Class

Object

Properties

Name	Type	Description
Attenuation	Integer	Specifies how the brightness of a point light falls off with distance. 0: No attenuation at all. 1: Brightness falls off linearly with distance. 2: Brightness falls off with the square of the distance. The default value is 1 (Linear).

Name	Type	Description
Brightness	Integer	Amount of brightness expressed as a percentage from 0 (completely off) to 100 (the default). Values greater than 100 are valid if you need extra brightness.
Direction	Vector3D	If Nil, you get a point light that radiates in all directions. If non-nil, it is the direction for a floodlight.
LightColor	Color	The color of the light. Defaults to white.
Position	Vector3D	If Nil, you get a floodlight. If non-nil, it is the position of a point light.

Notes

It is invalid to use a light with Position and Direction either both Nil, or both non-nil.

If the Attenuation value is 1 or 2, the effect of the light decreases with distance; so you may need to increase the Brightness substantially or make your objects smaller and closer to the light.

See Also

RB3DSpace, Vector3D, Bounds3D classes.

LightBevelColor Function

The currently selected Appearance Manager color for drawing light lines in dividing lines and group boxes.

Syntax

***result*=LightBevelColor**

Part	Type	Description
result	Color	The color used for drawing the light lines in dividing lines and group boxes.

Notes

This value is useful when you are using Canvas controls to create custom controls that you want to be Appearance Manager savvy. When drawing controls like dividing lines and GroupBoxes, use this color for the light portions of the object (usually the top and left sides of the object).

This value can be changed by the user on the fly using the Kaleidoscope Control Panel, so you should access this value during your Paint event handler rather than storing the value.

See Also

DarkBevelColor, DarkTingeColor, FillColor, FrameColor, HighlightColor, LightTingeColor, TextColor functions; Color data type.

LightTingeColor Function

The currently selected Appearance Manager color for drawing light lines inside frames on the top and left sides.

Syntax

***result*=LightTingeColor**

Part	Type	Description
result	Color	The color used for drawing the light lines inside frames.

Notes

This value is useful when you are using Canvas controls to create custom controls that you want to be Appearance Manager savvy. When drawing beveled objects, use this color for the light portions of the object (usually the top and left sides of the object). In a Checkbox control, this color is used to draw the light lines that give it the beveled appearance just inside the checkbox on the top and left sides.

This value can be changed by the user on the fly using the Kaleidoscope Control Panel, so you should access this value during your Paint event handler rather than storing the value.

See Also

DarkBevelColor, DarkTingeColor, FillColor, FrameColor, LightBevelColor, TextColor functions Color data type.

Line Control 

Draws a line.

Super Class

Control

Because this is a Control, see the Control Class for other properties and events that are common to all Control objects.

Properties

Name	Type	Description
BorderWidth	Integer	The width of the line in pixels.
LineColor	Color	The color of the line.
Name	String	The name of the object.
Super	Object Class	The class of object the object is based on.
Visible	Boolean	If True, the line will be visible when the window opens.

Name	Type	Description
X1	Integer	The point on the X axis where the line begins.
X2	Integer	The point on the X axis where the line ends.
Y1	Integer	The point on the Y axis where the line begins.
Y2	Integer	The point on the Y axis where the line ends.

Events

Name	Parameters	Description
MouseDown(function)	x as Integer, y as Integer	The mouse button was pressed inside the Line at the location passed in to x,y. Return True if you are going to handle the MouseDown.
MouseDrag	x as Integer, y as Integer	The mouse button was pressed inside the LIne and moved (dragged) at the location local to the control passed in to x,y. This event will not occur unless you return True in the MouseDown event.
MouseUp	x as Integer, y as Integer	The mouse button was released inside the Line region at the location passed in to x,y. This event will not occur unless you return True in the MouseDown event.

See Also

Control class.

ListBox Control

The scrollable **ListBox** control, used to display one or more columns of information. You can add a checkbox and/or a picture to a row. You can control font style on a cell-by-cell basis and set the column alignment. With some programming, you can create hierarchical lists that, for example, emulate the Macintosh's Finder List view.

The Controls Palette contains icons for both single column and multi-column List-Boxes. After using either icon, you can change the number of columns by setting the ColumnCount property. The only difference is that the multi-column ListBox tool lets you begin with a two-column ListBox.

Super Class

RectControl

Because this is a RectControl, see the RectControl for other properties and events that are common to all RectControl objects.

Properties

Name	Type	Description
ActiveCell		EditField that the ListBox uses for its editable cell operations. You can use this property to set or get the text of the Listbox cell (selText), set the selection, or change other properties of the ListBox's EditField.
Bold	Boolean	Applies the bold style to all items in the list.
CellAlignment	Integer	Parameters are row, column (Integers). The first cell is 0,0. Aligns the specified cell. Values are: 0 - Default alignment 1 - Left 2 - Center 3 - Right 4 - Decimal Decimal aligns the decimal separator to the right edge of the cell. You need to use CellAlignmentOffset or ColumnAlignmentOffset to move the data into the column.
CellAlignmentOffset	Integer	Parameters are row, column (Integers). Value is distance in pixels from the right edge of the cell. Overrides ColumnAlignmentOffset for that cell.
CellBorderBottom	Integer	Parameters are row, column (Integers).Sets the bottom border of the cell designated by *Row*, *Column* to a rule style. Possible values are: 0=Default 1=None 2=Thin Dotted 3=Thin Solid 4=Thick Solid 5=Double Thin Solid
CellBorderLeft	Integer	Parameters are row, column (Integers).Sets the left border of the cell designated by *Row*, *Column* to a rule style. Possible values are: 0=Default 1=None 2=Thin Dotted 3=Thin Solid 4=Thick Solid 5=Double Thin Solid

Name	Type	Description
CellBorderRight	Integer	Parameters are row, column (Integers).Sets the right border of the cell designated by *Row*, *Column* to a rule style. Possible values are: 0=Default 1=None 2=Thin Dotted 3=Thin Solid 4=Thick Solid 5=Double Thin Solid
CellBorderTop	Integer	Parameters are row, column (Integers).Sets the top border of the cell designated by *Row*, *Column* to a rule style. Possible values are: 0=Default 1=None 2=Thin Dotted 3=Thin Solid 4=Thick Solid 5=Double Thin Solid
CellCheck	Boolean	Parameters are row, column (Integers). The first cell is 0,0. Setting CellCheck to True checks the checkbox.
CellType	Integer	Parameters are row, column (integers). The first cell is 0,0. Values are: 0 - default 1 - normal 2 - add checkbox 3 - inline editable The value of CellType overrides ColumnType. For example, if ColumnType is 2, but a cell in the column has CellType set to 0, the cell will be normal.
Column	ListColumn	Parameter is *columnNumber* as Integer. The first column is numbered zero. Use -1 to refer to all columns. Enables you to access the ListColumn properties of the specified column.
ColumnAlignment	Integer	Parameter is column number; the first column is numbered zero. Aligns the specified column. Values are: 0 - Default alignment 1 - Left 2 - Center 3 - Right 4 - Decimal Decimal aligns the decimal separator to the right edge of the column. You need to use ColumnAlignmentOffset to move the data into the column.

Name	Type	Description
ColumnAlignmentOffset	Integer	Parameter is column number; the first column is numbered zero. Value is distance in pixels from the right edge of the column. A negative value moves the decimal separator to the left, i.e., into the body of the column. See the example in the Notes subsection "Decimal Alignment."
ColumnCount	Integer	The number of columns the ListBox contains. The maximum number of visible columns is 64 (columns 0 through 63).
ColumnSortDirection	Integer	Parameter is column number (Integer). The first column is numbered zero. Used to get or set the sort direction, according to the following values: -1 - Descending 0 - Don't sort 1 - Ascending (default) ColumnSortDirection doesn't actually sort the rows; it only establishes the sort direction that is used when the Sort method is called. The sort direction can be set even if there is no header for the ListBox. If you set ColumnSortDirection to Don't Sort, the user can't sort the column by clicking its header. Don't Sort will not call the SortColumns and CompareRows events.
ColumnType	Integer	Parameter is column number; the first column is numbered zero. Values are: 0 - default 1 - normal 2 - add checkbox 3 - inline editable (Disclosure triangles don't work in hierarchical ListBoxes if Inline Editable is selected.)

Name	Type	Description
ColumnWidths	String	A list of comma separated values, with each value controlling the width of the associated column. A value can be an absolute value (in pixels), a percentage, or a relative length expressed as *i** where *i* is an integer. If you use percentages, you can use non-integer values to specify fractions of a percent, e.g., 43.52%. The percentage value can be greater than 100%. If you use pixels, the last column doesn't grow to the size of the rest of the ListBox. Without any column width specifications, the headers will be divided evenly. If there are fewer column widths specified than the total number of columns, the remaining columns will divide up the remaining width equally. An element with a length of "3*" will be allotted three times the space of an element with length "1*". The value "*" is equivalent to "1*" and can be used to mean "fill the remaining space." You can use a mixture of pixels, percentages, and relative lengths. Column widths specified in pixels are guaranteed to have the specified width. Column widths specified in percentages are guaranteed to have that percentage of the visible width of the ListBox. The column widths specified using the * divide up the remaining width proportionally. For example, if there are four columns, the specification 20, 20%, *, 2* gives 20 pixels to the first column 20% of the total to the second column, and the last two columns divide up the remaining width in the ratio of 1:2, with the last column getting any remaining fractional pixels. Unrecognized or unsupported expressions (e.g. '2@')for the column width properties will result in an UnsupportedFormatException. The message of this exception describes in English what went wrong. Resizing a column will resize the value of the expression. If you resize the ListBox, both the percentage and relative lengths recompute their actual widths. There are two resizing "modes"; see notes. Header End caps are added for any additional unused space if headers are used. Header end caps do nothing when clicked.

Name	Type	Description
DataField	String	Relevant only if the ListBox is used in conjunction with a DataControl to display the contents of fields in a database table. Name of a field in the table referenced by DataControl.
DataSource	DataControl	The DataControl that references a table in a database whose fields are displayed using the ListBox. Use the DataField property to link the field to be displayed to the ListBox. In the IDE, a popup menu of field names will appear. When set in code, it takes a String, e.g., ListBox1.DataSource=DataControl1.Name
DefaultRowHeight	Integer	Determines the height of every row in the ListBox (pixels). To allow REALbasic to determine the height automatically based on the current font size, set DefaultRowHeight to -1.
EnableDrag	Boolean	Allows rows to be dragged.
EnableDragReorder	Boolean	If True, you can reorder rows within the ListBox by dragging rows. An insertion line indicator appears when dragging within the ListBox to provide you with visual feedback as to where the row would be dropped if you release the mouse button. Automatic drag reordering is not supported for hierarchical ListBoxes.
Expanded	Integer	Used to get or set the expanded state of the row passed. The row must have been added with the AddFolder method.
GridLinesHorizontal	Integer	Draws horizontal rules between rows in one of five styles: 0=Default (which is the default) 1=None 2=Thin Dotted 3=Thin Solid 4=Thick Solid 5 Double Thin Solid
GridLInesVertical	Integer	Draws vertical rules between columns in one of five styles: 0=Default (which is the default) 1=None 2=Thin Dotted 3=Thin Solid 4=Thick Solid 5 Double Thin Solid
HasHeading	Boolean	If True, a row of column headers is added to the ListBox. The user can sort the column by clicking the heading.

Name	Type	Description
Heading	String array	Zero-based array of column headings. Headings appear only if HasHeading is True. If you assign values to both Heading and InitialValue, the first row of InitialValue is interpreted as the first row of data; otherwise, it is used as the heading and the second row of InitialValue is used as the first row of data. You can set the headings in a multi-column listbox by assigning to Heading(-1) the text of the headings separated by the tab character, e.g., me.heading(-1)= "FirstName"+Chr(9)+"LastName" You must use a space if you want an empty header; empty strings will use the default header. ListBox1.heading(-1) = " " sets all headers of ListBox1 to their defaults. ListBox1.heading(5) = " " sets column 5's heading to its default heading ListBox1.heading(5) = " " sets column 5's heading to empty.
HeadingIndex	Integer	Allows you to get and set the sort column in a ListBox. The first column is numbered zero. To sort on a column, use ColumnSortDirection to set the sort direction to either ascending or descending and HeadingIndex to set the sort column. Then call the Sort method to do the sort.
Hierarchical	Boolean	Applies a Finder list-style background and allows for disclosure triangles for rows added via the AddFolder method. On Windows, plus and minus signs are used instead of disclosure triangles.
InitialValue	String	A list of the default items separated by carriage returns. If the ListBox has more than one column (ColumnCount > 1), separate column entries with Tabs within each row. If HasHeading is True, the first row of InitialValue is assumed to be the column headings—unless the Heading array is also specified. This property is unreadable at runtime because the list is removed from memory once the control is created.
Italic	Boolean	Applies the italic style to all items in the list.
LastIndex	Integer	The number of the last row added with the AddRow, AddFolder, or InsertRow method.
List	Array as String	The list of items. It takes one parameter, the row index. Use it to get or set the contents of that cell, or, in multi-column ListBoxes, the first column.
ListCount	Integer	The number of items in the list.
ListIndex	Integer	The selected item number.

Name	Type	Description
ScrollBarHorizontal	Boolean	If True, adds a horizontal scroll bar to the ListBox. The position of the thumb is indicated by ScrollPositionX. Used to scroll the ListBox horizontally without a separate ScrollBar control.
ScrollBarVertical	Boolean	If True, adds a vertical scroll bar to the ListBox. The position of the thumb is indicated by ScrollPosition.
ScrollPosition	Integer	Zero-based index of the top visible row in the ListBox. Read ScrollPosition to determine the top visible row; write to ScrollPosition to scroll the ListBox. When the scrollbar thumb is scrolled to the bottom, ScrollPosition cannot be incremented any further.
ScrollPositionX	Integer	Zero-based index of the horizontal position of the ListBox. Used with a ScrollBar control to scroll the ListBox horizontally. See the Notes section.
SelCount	Integer	The number of rows highlighted (selected).
Selected	Row as Integer	Indicates if the row passed is selected or not. This property can be used to determine if the row is selected and to select the row.
SelectionType	Integer	Indicates the type of selection allowed. 0=single row selection, 1=multiple row selection.
SortedColumn	Integer	Indicates at runtime which column is the sort column.
Text	String	The text of the currently selected item.
TextFont	String	Name of the font used to display the list text.
TextSize	Integer	Size of the font used to display the list text.
Underline	Boolean	Applies the underline style to all items in the list.
UseFocusRing	Boolean	If True, the object indicates that it has the focus with a ring around its border; if False, the appearance of the object does not change when it has the focus.

Events

Name	Parameters	Description
CellAction	Row as Integer, Column as Integer	If a cell is Editable, a CellAction event occurs when the user edits the cell. “Editing” is defined as exiting the cell after clicking in it. Clicking a checkbox in a checkbox cell also qualifies as “Editing.” The user doesn't necessarily have to change the contents.

Name	Parameters	Description
CellBackgroundPaint	g as Graphics Row as Integer Column as Integer	The parameter g is a Graphics object that corresponds to the content area of the cell *Row*, *Column*. 0,0 is the upper left of the cell. Returns a Boolean. True means the user has handled the background paint and no other processing is to be done with the background. In this case the user is responsible for all highlighting. False means the user wants REALbasic to help paint the background. REALbasic will attempt to highlight the row or column as appropriate (according to the platform (e.g., Win32, Mac OS 9, Mac OS X), and the hierarchical style).
CellClick	Row as Integer Column as Integer X as Integer Y as Integer	The user has clicked on the *Row*, *Column* cell. The parameters *X* and *Y* are the x and y coordinates of the mouse click relative to the top-left corner of the cell that was clicked. *X* and *Y* are on the same scale of reference as the coordinates used by the Graphics property of the CellBackgroundPaint event. Returns a Boolean. Returning True means that the event will not be processed further (i.e., editable cells won't be editable and ListBox selection won't change).
CellGotFocus	Row as Integer Column as Integer	The user has selected an editable cell of a ListBox. The *Row* and *Column* parameters indicate which cell just got the focus.
CellKeyDown	Row as Integer Column as Integer Key as String	The user has pressed a key while a cell in the ListBox is being edited. This cell is identified by the Row and Column parameters. Key is the key that the user pressed. Returns a Boolean. Returning True prevents the text from changing automatically and prevents the CellTextChange event from firing.
CellLostFocus	Row as Integer Column as Integer	The *Row*, *Column* cell has just lost the focus.The user could have clicked on another cell or pressed Return or Escape.
CellTextChange	Row as Integer Column as Integer	Occurs after the Keydown event if the KeyDown event returns False. The event passes the *Row* and *Column* of the cell being edited.

Name	Parameters	Description
CellTextPaint	g as Graphics x as Integer y as Integer Row as Integer Column as Integer	The parameter g is a Graphics object that corresponds to the text drawing area of the cell *Row*, *Column*. This does not necessarily correspond to the entire cell content area, for example, if you use a row picture in the cell. The parameters *x* and *y* are the coordinates of the suggested ideal location to draw text based on the current value of ColumnAlignment or CellAlignment, as well as the cell's font, font size, and font style. Returns a Boolean. The drawing order of the cell is as follows, with the background first: Background Disclosure Triangle/Treebox Checkbox RowPicture Text Border Although the border is painted last, it isn't advisable to change the state of the border in the CellTextPaint event since the area is determined by the size of the border before the cell is painted and so it could leave unpainted areas, or possibly cover up some of the painting you have done. True means the user has handled the text paint and no other processing is to be done with the text. In this case, the user is responsible for text highlighting. Text highlighting is currently only done for the hierarchical listbox style. False means the user wants REALbasic to paint the text. REALbasic will attempt to highlight the text as appropriate (according to the platform, and the hierarchical style).
Change		The selected item has changed.
CollapseRow	Row as Integer	The user has clicked on the disclosure triangle of the expanded Row passed.

Name	Parameters	Description
CompareRows	Row1 as Integer Row2 as Integer Column as Integer ByRef Result as Integer	Returns a Boolean. The CompareRows event is useful for sorting a column of a ListBox in a manner that is not provided by the default mechanism. The default mechanism sorts lexicographically. If you use the event, it gets called during a ListBox sort, e.g., when a user clicks in the header area. Parameters: Row1: Row number of one of the rows being compared. Row2: Row number of the other row being compared. Column: Number of column being sorted. Result: 0 — if items in Row1 and Row2 in specified column are equal. 1 — Contents of Row1 > Contents of Row2 -1 — Contents of Row2 > Contents of Row1 Return True if the returned Result parameter is accurate for sorting. Return False if you want REALbasic to use the default lexicographic sorting of the column.
DoubleClick		The user has double-clicked on an item.
DragReorderRows	NewPosition as Integer	Triggered when a row being reordered by dragging is dropped on the ListBox. The parameter *newPosition* indicates the row at which the dragged row is to be inserted. Returns a Boolean. Returning True prevents the reordering from happening. Automatic drag reordering is not supported for hierarchical ListBoxes. The DragReorderRows event fires, but the user is responsible for reordering the rows. It is the same behavior as returning True in the DragReorderRows event.
DragRow	Drag as DragItem, Row as Integer	The user is dragging a row. Drag is the DragItem object created automatically. Assign the values to the DragItem's properties that the user should drag. Row is the row of the ListBox that is being dragged. You must return True in this event handler to allow the drag to occur.
ExpandRow	Row as Integer	The user has clicked on the disclosure triangle of the collapsed Row passed.
GotFocus		The ListBox has the focus, i.e., the user has selected the ListBox.

Name	Parameters	Description
HeaderPressed	Column as Integer	Returns a Boolean. Runs after a ListBox header has been clicked/pressed. This event enables you to specify whether you want REALbasic to sort the column. If you return True, REALbasic does not sort the column and it does not update the SortedColumn property.
KeyDown	Key as String	The user has pressed the Key passed while the ListBox has the focus. Returns a Boolean. Returning True means that no further processing is to be done with the *Key*.
LostFocus		The ListBox has lost the focus.
MouseDown	x as Integer, y as Integer	Returns a Boolean. The mouse button was pressed inside the ListBox region at the location passed in to x,y. Returning True in the MouseDown event causes the MouseDrag and MouseUp events to fire and the default ListBox click processing not to fire.
MouseDrag	x as Integer, y as Integer	The mouse button was pressed inside the ListBox and moved (dragged) at the location local to the control passed in to x,y. This event will not occur unless you return True in the MouseDown event.
MouseUp	x as Integer, y as Integer	The mouse button was released inside the ListBox region at the location passed in to x,y. This event will not occur unless you return True in the MouseDown event.
SortedColumn	Column	The user has clicked on the column header to sort that column. Return True if you don't want the ListBox to sort.

Methods

Name	Parameters	Description
AddFolder	Item as String	Appends *Item* in a new row to the end of the list and adds disclosure triangle only if Hierarchical property is set to True. For multi-column ListBoxes, *Item* is always assigned to column zero. In the case of hierarchical ListBoxes, AddFolder appends *Item* to the subitems of the expanded row when called in the ExpandRow event.
AddRow	Item as String	Appends *Item* in a new row to the end of the list. For multi-column ListBoxes, *Item* is always assigned to column zero. In the case of hierarchical ListBoxes, AddRow appends *Item* to the subitems of the expanded row when called in the ExpandRow event.

Name	Parameters	Description
Cell	RowNumber as Integer, ColumnNumber as Integer	Used to read from or write to the cell based on the row and column numbers passed.
CellBold	RowNumber as Integer, ColumnNumber as Integer	Used to add or remove the bold style from the text of the specified cell. Assign True to add bold style and False to remove bold style.
CellItalic	RowNumber as Integer, ColumnNumber as Integer	Used to add or remove the italic style from the text of the specified cell. Assign True to add italic style and False to remove italic style.
CellUnderline	RowNumber as Integer, ColumnNumber as Integer	Used to add or remove the underline style from the text of the specified cell. Assign True to add underline style and False to remove underline style.
ColumnValueProvider	Column as Integer	Used in programming custom bindings. Returns StringProvider. Return object also implements DataNotifier and DataAvailableProvider.
DeleteAllRows		Deletes all rows in the list.
EditCell	Row as Integer, Column as Integer	Scrolls the *Row*, *Column* cell into view (if necessary) and temporarily makes the cell editable. It sets the focus within the ListBox to the *Row*, *Column* cell. See the CellGotFocus event. The editable cell has a focus ring around it.
InsertFolder	RowNumber as Integer; Item as String	Creates a new folder at *RowNumber* (moving the existing rows down). For multicolumn ListBoxes, *Item* is always assigned to column zero.
InsertRow	RowNumber as Integer, Item as String	Creates a new row at *RowNumber* (moving the existing rows down). For multicolumn ListBoxes, *Item* is always assigned to column zero.
PressHeader	Column as Integer	Causes the ListBox header to be pressed, causing a HeaderPressed event to occur. Calling this method does not update the sort direction.
RemoveRow	RowNumber as Integer	Deletes the specified row.
RowPicture	RowNumber as Integer	Adds the picture assigned to the RowNumber passed.
SetFocus		Gives the ListBox the focus sending all keydown events to the ListBox.

Name	Parameters	Description
Sort		Sorts the rows based on the current HeadingIndex and ColumnSortDirection. Although HeadingIndex also does this, this method is recommended. HeadingIndex may no longer sort rows in future releases of REALbasic.

Notes

Items in single-column ListBoxes can be accessed using the List property. The List property is an array. Arrays are zero-based which means that the first row of the List property of a **ListBox** is row number 0 (zero).

Multi-Column ListBoxes

You can create multi-column ListBoxes by changing the ColumnCount property. The first column in a multi-column **ListBox** is column 0 (zero). This means that the ColumnCount property will always be one more than the number of the last column. The maximum number of visible columns is 64 (columns 0 through 63). You should set ColumnCount to the number of columns that you want to display. If you want to put data in an invisible column, set the column width to zero.

You can use the InitialValue property to set up the inital values of multi-column ListBoxes by separating the column values with tabs and row values with carriage returns.

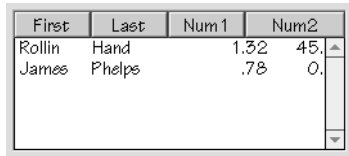
The widths of columns in multi-column ListBoxes can be set by passing the widths as a list of values separated by commas to the ColumnWidths property. The widths can be passed in pixels or as percentages of the total width of the **ListBox**. If you don't pass widths for all the columns, the remaining columns will be evenly spaced over the remaining space. If too many widths are passed, the additional values are ignored. If the total of the widths passed is greater than the width of the **ListBox**, then the remaining columns will be truncated.

Specific cells in a multi-column **ListBox** can be accessed using the Cell method.

Decimal Alignment

When you use decimal alignment in a cell or column, you must take into account the fact that REALbasic aligns the decimal separator with the right edge of the column or cell. You must pass a negative number to CellAlignmentOffset or ColumnAlignmentOffset to make room for the numbers to the right of the decimal place. The correct value to pass depends on the number of digits to the right of the decimal place in the column or cell.

The following illustrates the problem and the solution.



Without ColumnAlignmentOffset

ColumnAlignmentOffset(2)=-32
ColumnAlignmentOffset(3)=-18

Hierarchical ListBoxes

Creating a simple hierarchical **ListBox** is more involved than a two-column **ListBox** because you must manage hiding and displaying the sublist data. A simple way to do this is to assign the sublists to a "hidden" column in the **ListBox** and toggle the display of that data when the user double-clicks on a "parent" element.

You create a row with a disclosure triangle using the AddFolder method (rather than the AddRow method) and then set the Hierarchical property to True. See the Example for a simple hierarchical **ListBox** with one level.

Resizing Columns

There are two "modes" for column resizing. There is no formal mode property. Rather, the "mode" is implicitly set according to whether every column width is specified as an absolute amount. If you specify all columns either in pixels or as a percentage, you will be using the second mode. If you use an asterisk or leave a column width blank, you will be using the first mode.

- A change to one column width affects the width of another column.

  If column i gets bigger, column i+1 gets smaller by the same amount. This mode is great when using a ListBox without a horizontal scrollbar. You turn this mode on when you have at least one column width that is blank, or specified using an asterisk (e.g. "", " ", "*", or "4*")

  NOTE: By design you can't resize the right edge of the last column in this mode. To resize the last column you need to resize the previous column.

- Each column width is independent and can grow or shrink on its own.

  You are responsible when the user does this, and you need to provide a horizontal scrollbar so that the user can get to the any headers that have been pushed out of view to the right. You enable this mode by making sure every column width is specified in terms of an absolute pixel width, or a percentage width (e.g. "20", or "35%"). If you use an asterisk or leave a column width blank, you will automatically be using the first mode.

  You can switch between mode 1 and 2 at runtime using the same criteria as above.

The ColumnWidths property (The only one that can be set in the IDE), is equivalent to the concatenation of all of the ColumnWidthExpressions.

ColumnWidthExpressions are strings and they can represent several different types of column width calculations: absolute pixels (e.g., "45"), percentages (e.g. "22.3%"), and asterisk widths (or blanks) (e.g. " ", "4*")

ColumnWidthExpressions retain their type even when a column is resized. This means that if you:

- Resize a window to which a ListBox is locked, it will grow or shrink. The columns grow or shrink as well if their expressions were *-based (unless you use "0*"), or percentage based (0%). If you want them to stay fixed, you need to express the ColumnWidthExpression as an absolute pixel value.
- If you resize a column by dragging it internally, it will recompute its percentage or asterisk value. This is so that you can, say, start with a two-column ListBox with no column widths specified (each column will take up half the space). Then drag one column to take up 3/4 of the space, then enlarge the ListBox, and now both column widths will enlarge so that their widths remain in a 3/4 to 1/4 ratio.

  Changing the pixel value of a column will not change its fundamental type, but will change the value of that type.

  Finally, if you want to create columns that won't get resized, change the UserResizable property for each of the columns in question. If you are using mode 1, you will need to change the UserResizable property for both the column and the one to its left.

Databases

A **ListBox** is often used to display the results of database queries. A **ListBox** can be bound to a DatabaseQuery control so that it displays the results of the SQL query passed to the DatabaseQuery control automatically. See the example of object binding in the chapter "Creating Databases with REALbasic" in the *User's Guide*. Otherwise, a **ListBox** can be populated with the results of a query programmatically. See the example for DatabaseField.

Scrolling a ListBox Horizontally

The ScrollBarHorizontal property automatically adds a horizontal scrollbar to the **ListBox**. The ScrollPositionX property can be used to get the position of the thumb or scroll the **ListBox** horizontally. This is the easiest way to scroll a **ListBox**.

You can also use the ScrollPositionX property in conjunction with a ScrollBar control to scroll contents of the **ListBox** horizontally. You place a ScrollBar control below the **ListBox**, as shown below:

This **ListBox** has nine columns, eight of which have a width of 100 pixels and one has a width of 50 pixels.

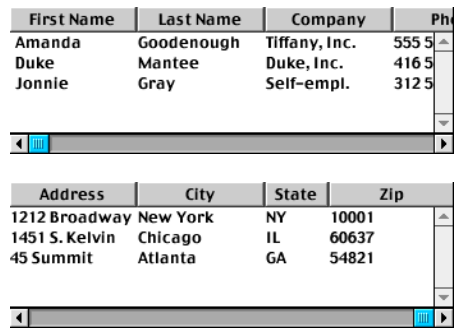
The ScrollBar control has the following code in its Open event handler:

```
Me.maximum=500
Me.minimum=0
Me.lineStep=50
```

The values for maximum and linestep were chosen to match the total width of the **ListBox**'s columns. Its ValueChanged event handler has the following line of code:

```
ListBox1.scrollPositionX=Me.value
```

n this way, the user can scroll the **ListBox** horizontally, bringing all columns into view.



Customized Scroll Controls

If you want miniature scrollbars or scrollbars that leave room for status bars or buttons, set ScrollBarVertical and/or ScrollBarHorizontal to False and use separate ScrollBar controls in conjunction with whatever other controls that you want to locate in the a portion of the area normally taken up by standard scrollbars. Use the ScrollPosition and ScrollPositionX properties to control scrolling, as described in the preceding section.

Horizontal and vertical rules

The following screen shots illustrate the four types of horizontal and vertical rules that can be drawn with the GridLinesHorizontal and GridLinesVertical properties and the CellBorderTop, CellBorderLeft, CellBorderRight, and CellBorderBottom methods.

The Default style is "None."

Thin Dotted

Thin Solid

Thick Solid

Double Thick Solid

With the CellBorderTop, …Left, …Bottom, and …Right methods, you can apply these ruling styles to selected cells or even selected cell borders.

For example, in this ListBox, a cell containing a phone number is highlighted using Thick Solid borders while the remainder of the ListBox uses Thin Dotted rules:

Examples

Adding a row to ListBox1:

```
ListBox1.addrow "October"
```

Inserting a row at row 1 in ListBox1:

```
ListBox1.insertRow 1, "October"
```

Changing all items in the ListBox to bold, underline:

```
ListBox1.bold=True
ListBox1.underline=True
```

Copying the fifth element of ListBox1 to another variable:

```
e=ListBox1.list(4)
```

Adding a column to ListBox1 and setting the widths of the columns to 50 and 65 pixels, respectively:

```
ListBox1.columnCount=2
ListBox1.columnWidths "50,65"
```

Adding two columns to ListBox1 and setting the widths of the columns to 60%, 20% and 20% respectively:

```
ListBox1.columnCount=3
ListBox1.columnWidths "60%,20%,20%"
```

If ListBox1 is 100 pixels wide and has three columns, the following code will set the columns widths but the last column will only be 10 pixels wide instead of 20:

```
ListBox1.columnWidths "60,30,20"
```

If ListBox1 is 100 pixels wide and has three columns, the following code will set the columns widths but the last column will not be displayed:

```
ListBox1.columnWidths "60,40,20"
```

Copying the fifth row of the third column of ListBox1 to another variable:

```
e=ListBox1.cell(4,2)
```

Assigning a value to the fifth row of the third column of ListBox1:

```
ListBox1.cell(4,2)="Bill"
```

Setting the fifth row of the third column of ListBox1 to bold, italic:

```
ListBox1.cellBold(4,2)=True
ListBox1.cellItalic(4,2)=True
```

Setting the picture for the fifth row of ListBox1 to “MyFolderPicture”:

```
ListBox1.RowPicture(4)=MyFolderPicture
```

Setting up the DragRow event handler to allow the user to drag a value from a ListBox:

```
Function DragRow(Drag as DragItem, Row as Integer) as Boolean
  Drag.Text=ListBox1.List(Row)
  Return True
End Function
```

Summing the numeric values of the selected rows:

```
Dim i, total as Integer
For i=0 to ListBox1.ListCount-1
  If ListBox1.Selected(i) then
    total=total+Val(ListBox1.List(i))
  End if
Next
```

This example expands the first row of ListBox1 (if it's collapsed) or collapses it (if it was expanded). The row must have been added with the AddFolder method:

```
ListBox1.Expanded(1)=Not ListBox1.Expanded(1)
```

This example populates a three-column ListBox with headings:

```
ListBox1.heading(0)="ID"
ListBox1.heading(1)="JobTitle"
ListBox1.heading(2)="Name"
```

This example sorts the first column in a **ListBox** and displays the number of the sort column in an EditField:

```
ListBox1.headingindex=0
EditField1.text=Str(ListBox1.headingindex)
```

This example sets up a **ListBox** with four visible columns plus one hidden column. Column zero is hidden:

```
Me.columncount=5
Me.columnwidths="0,25%,25%,25%,25%"
Me.heading(0)="ID"
Me.heading(1)="FirstName"
Me.heading(2)="LastName"
Me.heading(3)="Phone"
Me.heading(4)="Zip"
```

The following line of code displays the value of the hidden column in the selected row:

```
MsgBox ListBox1.Cell(ListBox1.ListIndex,0)
```

Hierarchical ListBoxes

The following example creates a single-level hierarchical ListBox that emulates the look and behaviour of Apple's Finder List view:

Collapsed State

An Expanded State

The following code, which is in the ListBox's Open event handler, populates the hierarchy: The s1 string contains the parent level and sub1 contains the elements that are nested within each of s1's elements. It is a list of comma-delimited lists, with each list delimited by semicolons.The elements of sub1 are initially hidden because they are stored in a hidden column.

```
Dim i, u as Integer
Dim s1,sub1 as string
Me.columnwidths="150,0"
s1="Michigan,Ohio,Minnesota"
sub1="Grand Blanc,Bad Axe,Flint,Benton Harbor,
Detroit;Cleveland,Columbus,Akron, Pleasantville; St. Paul,Frostbite Falls"
u=CountFields(s1,",")
For i=1 to u
 If NthField(sub1,";",i)<> "" then
  Me.addfolder ""
  Me.cell(i-1,1)=NthField(sub1,";",i)
 end if
 Me.cell(i-1,0)=NthField(s1,",",i)
Next
Me.ColumnCount=1
```

Note that the AddFolder method, rather than AddRow, is used to add the State names. Note that you must also set the Hierarchical property to True to display the disclosure triangles.

The following line of code in the Double-click event handler toggles the expanded state of the row that was double-clicked:

```
Me.expanded(Me.listindex)=Not Me.expanded(Me.listindex)
```

The following code in the ExpandRow event handler runs when the user double-clicks a collapsed element:

```
Dim s1 as string
Dim i,u as Integer
s1=me.cell(row,1)
u=CountFields(s1,",")
For i=1 to u
 Me.addrow ""
 Me.cell(me.lastIndex,0)=NthField(s1,",",i)
Next
```

It creates the sublist rows each time the user double-clicks a collapsed state name.

The following code is in the CollapseRow event handler:

```
Dim i,u,NSubRows as Integer
NSubRows=CountFields(Me.cell(row,1),",")
u=row+1
For i=row+NSubRows downto u
 Me.removerow i
Next
```

It removes the rows that were created by the ExpandRow event handler.

Drag and Drop between ListBoxes

The following example allows the user to drag one or more rows from ListBox1 to ListBox2. The dragged rows are added to the end of the list.

ListBox1 has its EnableDrag property set to True, enabling items in its list to be dragged, and its SelectionType property set to 1 (Multiple row selection). Its DragRow event handler is as follows:

```
Function DragRow (Drag as DragItem, Row as Integer)
 Dim nRows, i as Integer
 nRows=Me.ListCount-1
 For i=0 to nRows
  If Me.Selected(i) then
   Drag.AddItem(0,0,20,4)
   Drag.Text=Me.List(i) //get text
  End if
 Next
 Return True //allow the drag
End Function
```

It uses the AddItem method of the DragItem to add an additional item to the DragItem each selected row. The DropObject event handler then cycles through all items to retrieve all dragged rows.

ListBox2 has the following line of code in its Open event handler. It permits it to receive dragged text.

```
Me.AcceptTextDrop
```

Its DropObject event handler checks to see if the dragged object is text; if it is, it adds a row to the end of the list and assigns the text property of the dragged object to the new row: It loops through all items in the DragItem until NextItem returns False.

```
Sub DropObject(obj as DragItem)
Do
 If Obj.TextAvailable then
  Me.AddRow(Obj.Text)
 End if
Loop until Not obj.NextItem
End Sub
```

You can also drag from ListBox1 to the desktop to get a text clipping or to another application that supports text drag and drop, such as BBEdit.

Colors

This example, which is placed in the CellBackgroundPaint event, assigns alternating colors to the rows in a ListBox:

```
If row Mod 2=0 then
 g.foreColor=RGB(158,238,255)
else
 g.foreColor=RGB(172,255,145)
end if
g.FillRect 0,0,g.width,g.height
```

Notes: The CellBackgroundPaint event passes the parameters g (Graphics), and the Row and Column numbers (Integer). You can assign a color by creating it in the Colors window and then dragging the color to the text insertion point in the lines that assign a color to the ForeColor property.

The following line in the CellTextPaint event tells REALbasic to draw the text in the preceding example in red:

```
g.foreColor=RGB(255,0,0)
```

The CellTextPaint event is passed the coordinates of the suggested starting position to draw text in the parameters x and y. You can use them in a call to the DrawString property to specify the string to draw in a particular cell:

```
If row=4 and column=1 then
 g.foreColor=RGB(255,0,0)
 g.DrawString "Payment Overdue!",x,y
end if
Return True
```

Custom Sorts

The following example uses the CompareRows event to sort columns of numbers. If you rely on default sorting, numbers are not sorted in numerical order, i.e., 2 is greater than 100 and less than 200.

```
Function CompareRows(row1 as Integer, row2 as Integer, column as Integer,
ByRef result as Integer) as Boolean
 If Val(Me.Cell(row1,column))> Val(Me.cell(row2,column)) then
  result=1
 else
  result=-1
 End if
 Return True  //tells RB to use the custom sort
End Function
```

With this code in place, the correct (numerical) sorting is done whenever the user clicks the header area. Test to be sure that the custom sort affects only the numerical columns.

To sort dates, store the TotalSeconds property of the Date in a column of width zero and sort the rows based on that column.

See Also

DatabaseQuery, EditField controls; RecordSet, DatabaseField, ListColumn classes.

ListColumn Class

Used to refer to a particular column in a ListBox via its Column property.

Super Class

Object

Properties

Name	Type	Description
MaxWidthActual	Integer	The maximum width of the column (pixels).
MaxWidthExpression	String	The maximum width of the column as a string expression that can include spaces and the percent sign.
MinWidthActual	Integer	The minimum width of the column (pixels).
MinWidthExpression	String	The minimum width of the column as a string expression that can include spaces and the percent sign.
UserResizable	Boolean	True if the column is resizable; the default is False.
WidthActual	Integer	The width of the column in pixels.
WidthExpression	String	Column width as a string expression that can include the percent sign, a decimal point, or blanks. Width can be specified as a value in pixels, a percentage, or a relative length. To obtain the absolute width of a column specified using percent or a relative length, call WidthActual.

Notes

User resizable ListBox columns are an option beginning with REALbasic 4.5. Use the UserResizable property to set the property, e.g.,

```
ListBox1.Column(1).UserResizable=False
```

To set column widths, you can use a mixture of pixels, percentages, and relative lengths. Column widths specified in pixels are guaranteed to have the specified width with the user resizes the ListBox or resizable columns. Column widths specified in percentages are guaranteed to have that percentage of the visible width of the ListBox. The column widths specified using the "*" divide up the remaining width proportionally. Resizing a column will resize the value of the expression. If you resize the ListBox, both the percentage and relative lengths recompute their actual widths. There are two resizing "modes"; for more information, see the section "Resizing Columns" on page 261 in the documentation for the ListBox control.

For example, if there are three columns, the specification:

```
Listbox1.Column(0).widthExpression = "3*"
Listbox1.Column(1).widthExpression = "*"
Listbox1.Column(2).widthExpression = "10"
```

allocates a fixed amount of space to Column(2). There remaining width is divided up into four segments and allocates it between Columns 0 and 1 in the ratio of 3 to 1. This means that Column(0) gets the width of the ListBox minus 10 times 3/4, Column(1) gets the width of the ListBox minus 10) times 1/4, and Column(2) gets 10 pixels. As the ListBox changes in size (due to a window resize, for example) columns 0 and 1 will change to reflect that growth, while column 2 will not.

Header End caps are added for any additional unused space if headers are used. Header end caps do nothing when clicked.

If the string expression passed to WidthExpression, MinWidthExpression, or MaxWidthExpression cannot be evaluated to a width or a percentage width, an UnsupportedFormatException will occur. The Message property of this exception describes in English what went wrong.

Example

The following line in the Open event of a ListBox makes all the columns in the ListBox resizable:

```
Me.Column(-1).UserResizable=True
```

The following line sets the maximum width of the first column in a ListBox to 75% of the total width of the ListBox.

```
Listbox1.Column(0).MaxWidthExpression="75%"
```

See Also

ListBox class.

ListSelectionNotificationReceiver Interface Class

Used to program custom object bindings. See the example of custom object bindings in the *User's Guide*.

Methods

Name	Parameters	Description
SelectionChanging		The selection is changing
SelectionChanged		The selection has changed

ListSelectionNotifier Interface Class

Used to program custom object bindings.

Methods

Name	Parameters	Description
addListSelectionNotificationReceiver	receiver as ListSelectionNotificationReceiver	Adds receiver object to custom interface class
removeListSelectionNotificationReceiver	receiver as ListSelectionNotificationReceiver	Removes receiver object from custom interface class.

LittleArrows Control

Adds a pair of vertical arrows to a window. You can, for example, use a LittleArrows control to increment or decrement the value of another object.

Super Class

RectControl

Because this is a RectControl, see the RectControl for other properties and events that are common to all RectControl objects.

Events

Name	Parameters	Description
Down		The user has clicked the Down arrow.
Up		The user has clicked the Up arrow.

See Also

RectControl class.

Log Function

Returns the natural logarithm of the value specified.

Syntax

***result*=Log (*value*)**

Part	Type	Description
result	Double	The natural logarithm of *value*.
value	Double	The value you want the natural logarithm of.

Examples

This example uses the **Log** function to return the natural logarithm of a number.

```
Dim d As Double
d=Log(10) //returns 2.3025851
```

Lowercase Function

Converts all characters in a string to lowercase characters.

Syntax

***result*=Lowercase(*value*)**

OR

***result=stringVariable*.Lowercase**

Part	Type	Description
result	String	A copy of the original string with all characters converted to their lowercase equivalent.
value	String	The original string.
stringVariable	String	Any variable of type String.

Notes

Returns the value with all alphabetic characters in lowercase.

Examples

The example below converts the value passed to lowercase.

```
Dim s As String
s=Lowercase("tHe Quick fOX") //returns "the quick fox"
s=Lowercase("THE 5 LAZY DOGS") //returns "the 5 lazy dogs"
```

See Also

Uppercase, Titlecase functions.

LTrim Function

Returns the string passed with leading (left side) spaces removed.

Syntax

***result*=LTrim(*SourceString*)**

OR

***result*=*stringVariable*.Ltrim**

Part	Type	Description
result	String	*SourceString* with leading spaces removed.
SourceString	String	The source, a copy of which, to be returned with leading spaces removed.
stringVariable	String	Any variable of type String.

Examples

This example removes the spaces from the left side of the string passed:

```
Dim s as String
s=LTrim("  Hello World  ")
//Returns "Hello World  "
```

See Also

RTrim, Trim, Asc, Chr, InStr, Len, Left, Mid, Right functions.

Max Function

Returns the larger of the two numbers passed to it.

Syntax

***result*=Max(*value1*, *value2*)**

Part	Type	Description
result	Double	The greater of *value1* and *value2*.
value1	Double	The first number for comparison.
value2	Double	The second number for comparison.

Notes

The **Max** function compares the two numbers and returns the greater of the two.

Examples

This example uses the **Max** function to return the greater of the two values passed.

```
Dim d As Double
d=Max(3.01, 4.05)   //returns 4.05
d=Max(3.012, 3.011) //returns 3.012
```

MD5 Function

Returns the MD5 message-digest value of a String.

Syntax

***result*=MD5(*String*)**

Part	Type	Description
Result	String	The MD5 message digest form of the passed String.
String	String	The input string to be parsed.

Notes

Use the **MD5** function to process the raw data to be processed. For complete information on the MD5 message digest format, see RFC 1321.

See Also

MD5Digest class.

MD5Digest Class

Processes a string and returns the message-digest form of the input string.

Super Class

Object

Properties

Name	Type	Description
Value	String	Contains the current message digest.

Methods

Name	Parameters	Description
Process	Data as String	Data is the text to be processed.
Clear		Resets the MD5Digest object so that you can start with a new data stream.

Notes

The **MD5Digest** class enables you to process a string in segments. Pass each string segment to the Process method. The Value property contains the current message digest and the Clear method clears the **MD5Digest** object so that you can repeat the process.

The MD5 message digest algorithm takes a message of any length and produces a 128-bit "fingerprint" or *message digest* of the input string. The MD5 algorithm is useful for digital signature applications, where a large file must be processed in a secure manner

before being encrypted with a secret key under a system such as RSA. See RFC 1321 for complete information.

See Also

MD5 function.

MDIWindow Class

Used to configure the MDI window on Multiple Document Interface built applications (Windows only).

Super Class

Application

Properties

Name	Type	Description
Handle	Integer	Returns handle to the MDI window. Any visible Win32 control should have a handle.
Height	Integer	The height of the MDI window.
Left	Integer	The distance (in pixels) between the left edge of the screen and the left edge of the MDI window.
MinHeight	Integer	The minimum height to which the MDI window can be resized.
MinWidth	Integer	The minimum width to which the MDI window can be resized.
MaxHeight	Integer	The maximum height to which the MDI window can be resized.
MaxWidth	Integer	The maximum width to which the MDI window can be resized.
Title	String	The MDI window title.
Top	Integer	The distance (in pixels) between the top edge of the screen and the top edge of the MDI window.
Visible	Boolean	If True, the MDI window will be visible when it's opened.
Width	Integer	The width of the MDI window.

Events

Name	Parameters	Description
Moved		Occurs when the MDI window moves.
Open		Occurs when the MDI window is created.
Resized		Occurs when the MDI window resizes.

Methods

Name	Parameters	Description
Maximize		Maximizes the MDI window.

Name	Parameters	Description
Minimize		Minimizes the MDI window.
Restore		Restores the MDI window.

Notes

In the Build Settings dialog box, you have the option of using the Multiple Document Interface (MDI) for a Windows build of your application. If you select this option, all of your application's windows will be enclosed in a "parent" window called the MDI window. If you don't select the MDI interface, your application's windows will be on the desktop, like a Macintosh application. This class allows you to set certain properties and behaviors of the MDI window.

Example

MDIWindow class is accessible directly from the Application class using the App object. The following code in the application's Open event sets some properties of the MDI window:

```
App.MDIWindow.Title="REAL Scheduler"
App.MDIWindow.MinHeight=225
App.MDIWindow.Height=72
```

See Also

Application class; App object.

Me Function

Me is a reference to an event handler's control.

Notes

When called within an event handler or method of a control on a form, **Me** will always be a reference to the control that owns the event handler or method where **Me** was called.

This means that in PushButton1's Action event handler:

PushButton1.Enabled=False and **Me**.Enabled=False

mean the same thing. When **Me** is called within the method of a class, **Me** will be a reference to the instance of the class in use. The **Me** function is different from the Self function in that Self always refers to the object's parent and not the object itself.

For code for a control in a window, "Self" refers to the window, and "**Me**" refers to the control.

One big advantage of using the **Me** function when referring to a control in one of its event handlers is that if you change the name of the control, the code in its own event

handlers will not need to be updated. Also, you can copy the code to another control of the same type and it will work without modification.

Example See the example for hierarchical ListBoxes.

See Also Self function.

Me Cannot be used in a Method of a Module Error

You can't use Self or Me in a method or function in a module because there is no parent window or control. Self can be used only when there is a parent window for the method and Me can be used only in a control's event handler.

See Also Self, Me function.

MemoryBlock Class

A **MemoryBlock** object allocates a sequence of bytes in memory and manipulates those bytes directly. MemoryBlock objects can be passed to CFM entry points, where the parameter has been declared as Ptr.

Super Class Object

Properties

Name	Parameters	Description
BooleanVaue	Offset as Integer	Get (0=False; ≠0 is True) or set (0=False; 1=True) a one-byte Boolean value. Returns a Boolean.
Byte	Offset as Integer	Returns an integer.
ColorValue	Offset as Integer, Bits as Integer	Get or set a Color in one of three formats, selected by *Bits*. 32: 32-bit color (high byte unused) 24: 24-bit color 16: 16-bit color (5 bits per color, high bit unused) Returns a Color.
CString	Offset as Integer	Returns a String. The last byte is $00.
DoubleValue	Offset as Integer	Gets or sets a Double value.

Name	Parameters	Description
LittleEndian		Used to set the lets you specify the endianness of a memoryblock. The default is the endianness of the platform on which the code is being compiled. It is a Boolean.
Long	Offset as Integer	Returns an Integer, four bytes in length.
PString	Offset as Integer	Returns a String, up to 255 characters. The first byte is the length of the string.
Ptr	Offset as Integer	Returns a **MemoryBlock**.
Short	Offset as Integer	Returns a signed 16 bit Integer.
SingleValue	Offset as Integer	Returns a 4-byte representation of a single-precision floating number as an Integer.
StringValue	Offset as Integer, Length as Integer	Gets and sets a String of arbitrary size; may contain nulls, and is not prefixed with a length byte. When setting, if *Length* is greater than the length of the string, it is padded with zeros.
Size		Returns the size of the **MemoryBlock** in bytes. If you assign a value to this property, REALbasic resizes the **MemoryBlock**, retaining as much of the existing data as possible. Returns an Integer.
UShort	Offset as Integer	Returns an unsigned 16 bit Integer.

Notes

Memoryblocks can be used whenever you need a container for any arbitrary binary data.

Memoryblocks are used when accessing entry points in shared libraries and when making OS calls using the Declare function. For more information on shared libraries, see Apple's *Inside Macintosh* series.

Examples

This is an example where a Gestalt function is being declared with the parameters of Integer, Ptr returning an Integer:

```
Function Gestalt(setting As String) As Integer
  Dim m as MemoryBlock
  Dim result as MemoryBlock
  Dim err as Integer

  m = NewMemoryBlock(10)
  result = NewMemoryBlock(4)
  m.CString(0) = setting

  err = InterfaceLib.Gestalt(m.long(0),result)

  Return result.long(0)
End Function
```

The following example reads a real number into a memory block and then displays it:

```
Dim m as MemoryBlock
Dim d as Single
m=NewMemoryBlock(4)
m.singlevalue(0)=123.456
d=m.singlevalue(0)
MsgBox Str(d)
```

The following example stores a string in a MemoryBlock and displays it:

```
Dim m as MemoryBlock
m=NewMemoryBlock(13)
m.byte(0)=12
m.byte(1)=72
m.byte(2)=101
m.byte(3)=108
m.byte(4)=108
m.byte(5)=111
m.byte(6)=32
m.byte(7)=87
m.byte(8)=111
m.byte(9)=114
m.byte(10)=108
m.byte(11)=100
m.byte(12)=33
MsgBox m.pstring(0)
```

To read the string using cString, add the terminating byte before the call, i.e.,

```
m.byte(13)=0
MsgBox m.cString(1)
```

See Also

NewMemoryBlock function.

MenuItem Class

An individual menu item added via the Menu Editor or added by default to the Edit menu. The Quit menu item is a special case; it is derived from the QuitMenuItem class.

Super Class Object

Properties

Name	Type	Description
AutoEnable	Boolean	If set to True, the menu item is enabled by default, as long as the Application object or frontmost window has a menu handler for it. There is no need to put code in the EnableMenuItems event handler to explicitly enable the menu item. AutoEnable is True by default.
BalloonHelp	Message as String	Message that appears when Balloon Help is on and the user holds the mouse over the menu item while the item is enabled.
Bold	Boolean	Indicates if the menu item should appear in bold.
Checked	Boolean	Indicates whether or not the menu item is checked.
CommandKey	String	The Macintosh keyboard shortcut for the menu item. Keyboard shortcuts for Windows are set using Constants. See the section "Using Constants to Add Keyboard Shortcuts to Menus and Menu Items" in the *User's Guide*.
DisabledBalloonHelp	Message as String	Message that appears when Balloon Help is on and the user holds the mouse over the menu item while the item is disabled.
Enabled	Boolean	Indicates whether or not the menu item is enabled. This property can be set only from an EnableMenuItems event handler. You can enable a menu item either by assigning the Enabled property a value of True or by calling the Enable method.
Index	Integer	The number of the MenuItem when it is part of an array.
Italic	Boolean	Indicates if the menu item should appear in italic.
Name	String	The name of the menu item.
Submenu	Boolean	Indicates that the menu item is a submenu.
Tag	String	A 'hidden' text string associated with the menu item. The tag is accessible via code when the user chooses the menu item but, unlike the Text property, is not displayed in the menu. It works like the RowTag property of a PopupMenu control.
Text	String	The text of the menu item.

Name	Type	Description
Underline	Boolean	Indicates if the menu item should appear underlined.
Visible	Boolean	Indicates whether or not the menu item is visible. This property is not supported on Windows.

Methods

Name	Parameters	Description
Append	newChild as **MenuItem**	Appends the passed **MenuItem** to the menu.
Child	Name as String	Looks up menu items by name and or by Text and returns a **MenuItem**.
Close		Removes dynamically created menu items.
Count		Returns as an Integer the number of menu items. For a menu item, it returns the number of submenu items, if any. If there are no submenu items, it returns zero.
Enable		Sets the Enabled property of the menu item to True. Call Enable only within an EnableMenuItems event handler.
Insert	Index as Integer newChild as **MenuItem**	Inserts *newChild* as a **MenuItem** at the position indicated by *Index*.
Item	Index as Integer	Item returns as a **MenuItem** the n^{th} item (zero-based). It throws an OutOfBoundsException if the supplied index is out of range.
Remove	Index as Integer OR Child as **MenuItem**	Removes the **MenuItem** specified either by its position (index) or by reference.

Notes

MenuItem objects are used to access the properties of menu items. You can change the menu item text using the Text property. You can find out if a menu item is checked, or check or uncheck a menu item with the Checked property. You can also enable or disable a menu item using the Enabled property. The Enabled property should be set only from within an EnableMenuitems event handler. Setting it from anywhere else has no effect.

Two other classes handle specialized menu items. QuitMenuItem is designed to manage the File ▶ Quit menu of a built application; it is enabled by default and automatically calls the Quit method. The PrefsMenuItem class is designed to handle the Preferences menu item. In Mac OS X, this menu item is supposed to be located under the application's menu, but on other operating systems, this menu does not exist. A menu item derived from the PrefsMenuItem class automatically appears under the application's

menu under Mac OS X; on other operating systems is appears where you put it in the Menu Editor.

MenuItems can be created on the fly using the New operator. See the New operator for more information.

Examples

The following example changes the text of the EditPaste menu item to "Paste Special…":

```
EditPaste.text="Paste Special..."
```

The following example removes the fourth dynamically created menu item from a menu item array named WindowItem:

```
WindowItem(3).Close
```

The following example assigns a value to the Tag property of a menu item:

```
SearchFind.Tag="UserSearch"
```

See Also

PrefsMenuItem, QuitMenuItem classes; EnableMenuItems function, New operator.

Microseconds Function

Returns the number of microseconds (1,000,000th of a second) that have passed since the user's computer was started.

Syntax

***result*=Microseconds**

Part	Type	Description
result	Double	The number of microseconds that have passed since the user's computer was started. On Windows only, the result is accurate only to milliseconds.

Examples

This example displays in message box the number of minutes the computer has been on.

```
Dim minutes As Integer
minutes=Microseconds/1000000/60
Msgbox "Your computer has been on for "+Str(minutes)+" minutes."
```

Mid Function

Returns a portion of a string. The first character is numbered 1.

Syntax

***result*=Mid(*source*, *start* [,*length*])**

OR

***result*=*stringVariable*.Mid(*start*,[*length*])**

Part	Type	Description
result	String	The portion of *source* from *start* and continuing for *length* characters or all remaining characters if *length* is not specified.
source	String	Required. The string from which characters are returned.
start	Integer	Required. The position of the first character to be returned. If *start* is greater than the number of characters in *source*, an empty string is returned.
length	Integer	Optional. The number of characters to return from *source*. If ommitted, all characters from *start* to the end of *source* are returned. If *length* + *start* is greater than the length of *source*, all characters from *start* to the end of *source* will be returned.
stringVariable	String	Any variable of type String.

Notes

To determine the number of characters in a string, use the Len function.

If you need to specify the length in bytes rather than characters, use the LenB function.

The **Mid** function works properly with international text.

Examples

These examples use the **Mid** function to return portions of a string.

```
Dim s As String
s = Mid("This is a test", 6) //returns "is a test"
s = Mid("This is a test", 11, 4) //returns "test"
```

This example converts the text string in EditField1 to hex and writes the result to EditField2:

```
Dim i as Integer
EditField2.text=""
For i=1 to LenB(EditField1.text)
 EditField2.text=EditField2.text+"\x"+Hex(Asc(Mid(EditField1.text,i,1)))
Next
```

See Also

Asc, Chr, InStr, Left, Len, Right functions.

MidB Function

Returns a portion of a string. The first character is numbered 1.

Syntax

***result*=MidB(*source*, *start* [,*length*])**

OR

***result*=stringVariable.MidB(*start*, [*length*])**

Part	Type	Description
result	String	The portion of *source* from *start* and continuing for *length* characters or all remaining characters if *length* is not specified.
source	String	Required. The string from which bytes are returned.
start	Integer	Required. The position of the first byte to be returned. If *start* is greater than the number of bytes in *source*, an empty string is returned.
length	Integer	Optional. The number of bytes to return from *source*. If omitted, all bytes from *start* to the end of *source* are returned. If *length* + *start* is greater than the length of *source*, all bytes from *start* to the end of *source* will be returned.
stringVariable	String	Any variable of type String.

Notes

MidB treats *source* as a series of bytes, rather than a series of characters. **MidB** should be used when *source* represents binary data. If you need to extract characters rather than bytes, use the Mid function. To determine the number of bytes in a string, use the LenB function.

Examples

These examples use the **MidB** function to return portions of a string.

```
Dim s As String
s=MidB("This is a test", 6) //returns "is a test"
s=MidB("This is a test", 11, 4) //returns "test"
```

See Also

AscB, ChrB, InStrB, LeftB, LenB, Mid, RightB functions.

Min Function

Returns the smaller of the two numbers specified.

Syntax

***result*=Min(*value1*, *value2*)**

Part	Type	Description
result	Double	The lesser of *value1* and *value2*.
value1	Double	The first number for comparison.
value2	Double	The second number for comparison.

Notes

The **Min** function compares the two numbers and returns the smaller of the two.

Examples

This example uses the **Min** function to return the lesser of the two values passed.

```
Dim d As Double
d=Min(3.01, 4.05)   //returns 3.01
d=Min(3.012, 3.011) //returns 3.011
```

Mod Operator

Returns the remainder of the division of two numbers.

Syntax

result*= *number1* Mod *number2

Part	Description
result	The remainder (as an Integer) of number1 divided by number2.
number1	Any number.
number2	Any number.

The **Mod** operator divides *number1* by *number2* and returns the remainder as *result*. If *number2* is zero, **Mod** returns *number1*.

The **Mod** operator operates on integers. If either *number1* or *number2* are non-integers, they are first truncated to integers.

For example:

5 **Mod** 2 returns 1
5 **Mod** 1.99999 returns 0

Examples

These examples use the **Mod** operator to divide two numbers and return the remainder.

```
Dim r As Integer
r=10 Mod 3 //Returns 1
r=2 Mod 4  //Returns 2
r=9.3 Mod 2.75 //Returns 1
r=4.5 Mod 1 //Returns 0
r=25 Mod 5 //Returns 0
```

See Also

/, \ operators.

MouseCursor Class

Used to specify the appearance of the pointer.

Super Class

Object

Notes

See the MouseCursor properties of the Application class, the Window class, and the Control class. The relationships among them is as follows:

The MouseCursor property of a control determines the shape of the mouse pointer when the pointer enters the control's region *only* if the Application and Window classes' MouseCursor properties are Nil. Similarly, the Window class's MouseCursor property determines the shape of the pointer when it enters the region of the window only if the Application class's MouseCursor property is Nil. If the Application class's MouseCursor property is not Nil, then it controls the shape of the pointer and a window's and any control's MouseCursor property values are ignored.

A MouseCursor can be assigned in three ways:

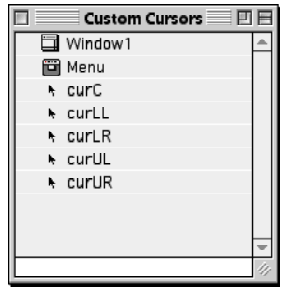
- You can use the names WatchCursor, IBeamCursor, or ArrowCursor.
- You can drag a resource file into the Project Window that contains a CURS resource and use the ResourceFork.GetCursor method to obtain a cursor resource.
- You can open a file's resource fork and open any CURS or crsr resource using the ResourceFork.GetCursor method.

Custom Cursors

You can add custom cursors using the following technique:

Create a separate CURS resource that contains only one cursor for each custom cursor you wish to use. Give each resource file a different name and drag them to the Project window.

When you do this, each resource file has a pointer icon:



You can access each cursor using the resource's name, e.g.,

```
Self.mousecursor=curC
```

See Also

MouseCursor properties of the Application class, the Window class, and the Control class.

Movie Class

A reference to a QuickTime movie. Movie Class objects have no events.

Super Class

Object

Properties

Name	Type	Description
BaseMovieHeight	Integer	Actual height of the movie (pixels).
BaseMovieWidth	Integer	Actual width of the movie (pixels).
Handle	Integer	Returns handle to movie, mainly used for toolbox calls that require the handle to a movie.
MovieHandle	Integer	Handle to movie. Provides the same functionality as Handle, but may not be supported in future releases of REALbasic. Obsolescent, use Handle instead.
MovieHeight	Integer	Height of the QuickTime movie in pixels
MovieWidth	Integer	Width of the QuickTime movie in pixels.

Notes

QuickTime movie objects can be accessed via the Movie property of MoviePlayer control or when calling the OpenAsMovie method of a FolderItem object.

Examples

This example opens a QuickTime movie on disk, assigns it to the movie property of a MoviePlayer object, and displays its height and width in EditFields.

```
Dim f as FolderItem
Dim m as Movie
f=GetFolderItem("Logo.mov")
m=f.OpenasMovie
moviePlayer1.border=True
moviePlayer1.movie=m
moviePlayer1.controller=2
EditField2.text=str(m.movieheight)
EditField3.text=str(m.moviewidth)
```

See Also

OpenURLMovie function; EditableMovie, QTTrack classes; MoviePlayer control.

MoviePlayer Control

Displays an area for playing QuickTime movies.

Super Class

RectControl

Because this is a RectControl, see the RectControl for other properties and events that are common to all RectControl objects.

Properties

Name	Type	Description
AutoResize	Boolean	If True, the MoviePlayer control resizes itself to fit the size of the movie.
Border	Boolean	Draws a border around the movie area.
Controller	Integer	0=no movie controls, 1=Badge, 2=Full controller
ControllerHeight	Integer	Height of the controller (pixels).
ControllerWidth	Integer	Width of the controller (pixels).
EditingEnabled	Boolean	If True, it is possible to edit the movie. A MoviePlayer's methods for editing a movie work when EditingEnabled is True.
HasStep	Boolean	If True, the controller has forward and reverse arrows to the right of the slider. When set in code, HasStep takes effect the *next* time the MoviePlayer is assigned a movie.
Looping	Boolean	Re-plays the movie automatically when it reaches the end.

Name	Type	Description
Movie	Movie	The QuickTime movie that will be played.
Palindrome	Boolean	Plays the movie in reverse when the movie reach its end. Looping must be True for Palindrome to work.
PlaySelection	Boolean	If a selection is made (via SelStart and SelLength) and this property is set to True, only the selection will be played. If False, normal playback will occur.
Position	Double	Number of seconds from the start of the movie.
QTMovieController	Integer	Returns a pointer that refers to the MoviePlayer control. Of use to toolbox programmers.
QTVRNode	Integer	The node that is currently being shown in the controller.
QTVRNodeCount	Integer	The number of nodes in the MoviePlayer's movie.
QTVRPan	Double	The pan angle in degrees of MoviePlayer's QTVR movie.
QTVRPanMax	Double	The maximum pan angle (in degrees) of the MoviePlayer's QTVR movie.
QTVRPanMin	Double	The minimum pan angle (in degrees) of the MoviePlayer's QTVR movie.
QTVRPanTiltSpeed	Integer	The speed of tilt motion (angle in degrees) of the MoviePlayer's QTVR movie.
QTVRTilt	Double	The tilt angle (in degrees) of the MoviePlayer's QTVR movie.
QTVRTiltMax	Double	The maximum tilt angle (in degrees) of MoviePlayer's QTVR movie.
QTVRTiltMin	Double	The minimum tilt angle (in degrees) of MoviePlayer's QTVR movie.
QTVRZoom	Double	The desired vertical field of view for the specified movie. This value is constrained by the maximum field of view of the movie. Values that lie outside that limit are clipped to the maximum. Pan and tilt angle values are also clipped if, when combined with the current field of view, they would cause an image to lie outside the current constraints.
QTVRZoomMax	Double	The maximum zoom field of view.
QTVRZoomMin	Double	The minimum zoom field of view.
QTVRZoomSpeed	Integer	The rate at which you can zoom.
Rate	Double	Sets the playback rate. For example, the following values have these effects: 0 —Paused / stopped 0.5 — Half normal playback rate 1 — Normal playback rate 2 — Double normal playback rate -1 — Reverse normal playback rate Since Rate is a Double, other values, including values outside this range, work.
SelLength	Double	Gets or sets the length (in seconds) of the selection.

Name	Type	Description
SelStart	Double	Gets or sets the starting position (in seconds) of the selection.
Speaker	Boolean	Displays the QuickTime speaker volume control. When set in code, Speaker takes effect the *next* time the MoviePlayer control is assigned a movie.
Volume	Integer	Gets or sets the movie's volume. The range is from 0 to 256.

Events

Name	Description
ControllerSizeChanged	The size of the controller has changed. When a movie first starts this event fires and allows you read the ControllerWidth/Height properties and then resize the MoviePlayer and or window to best accommodate that size. The event also fires if the movie changes size 'on its own' (QT4 only).
Play	The current movie is about to begin playing.
Stop	The current movie is about to stop playing.

Methods

Name	Parameters	Description
Clear		Clears the movie's current selection. Set the EditingEnabled property to True to enable the user to make a selection using the MoviePlayer's controller. If the movie is not an EditableMovie, no changes can be saved.
Copy		Copies the movie's current selection to the Clipboard. Set the EditingEnabled property to True to enable the user to make a selection using the MoviePlayer's controller. If the movie is not an EditableMovie, no changes can be saved.
Cut		Cuts the movie's current selection and places it on the Clipboard. Set the EditingEnabled property to True to enable the user to make a selection using the MoviePlayer's controller. If the movie is not an EditableMovie, no changes can be saved.
Paste		Pastes the contents of the Clipboard to the Movie's current selection. Set the EditingEnabled property to True to enable the user to make a selection using the MoviePlayer's controller. If the movie is not an EditableMovie, no changes can be saved.
Play		Plays the current movie.

Name	Parameters	Description
QTVRHotSpotCount	nodeID As Integer	Returns an Integer, the number of QTVR hot spots in the movie with the node passed in nodeID.
QTVRHotSpotID	node As Integer, index as Integer	Returns an Integer, the ID of a specific hot spot based on node number and hot spot index passed as parameters.
QTVRNodeTypeObject	node As Integer	Determines whether the current movie has a QTVR object in it. Returns a Boolean; True if the movie has a QTVR object.
QTVRNodeTypePanorama	node as Integer	Determines whether the current movie has a QTVR Panorama in it. Returns a Boolean; True if the movie has a QTVR Panorama.
QTVRToggleHotSpotNames		Toggles whether HotSpot names are shown in the movie controller.
QTVRTriggerHotSpot	hotSpotID as Integer	Navigates to the specified hot spot in the MoviePlayer's QTVR movie.
Stop		Stops the current movie.
Trim		Retains the current selection (specified by SelStart and SelLength) and removes all other media. Set the EditingEnabled property to True to enable the user to make a selection using the MoviePlayer's controller. If the movie is not an EditableMovie, no changes can be saved.
Undo		Undo the last operation. This is actually an Undo/Redo toggle. Set the EditingEnabled property to True to enable the user to make a selection using the MoviePlayer's controller. If the movie is not an EditableMovie, no changes can be saved.

Notes

The Controller property dictates how the QuickTime movies controls (if any) will be displayed. Passing 0 (zero) means that there will be no user controls available. Passing 1 means that a movie icon or badge will be displayed in the lower left corner of the movie area instead of the controller. When this badge is clicked by the user, the badge disappears and the regular QuickTime movie controls appear at the bottom of the movie frame. Passing 2 displays the regular QuickTime movie controls.

Examples

This example sets the QuickTime movie "MacintoshJr" as the movie to be played.

```
MoviePlayer1.movie=MacintoshJr
```

This example loads a QuickTime movie called "MyMovie" from the current directory (folder) into MoviePlayer1 and plays it.

```
Dim f As FolderItem
f=New FolderItem
f=GetFolderItem("MyMovie")
MoviePlayer1.Speaker=True
MoviePlayer1.HasStep=False
MoviePlayer1.movie=f.OpenAsMovie
MoviePlayer1.play
```

If you dragged the movie into your Project window, you can assign it to the movie property with only one line of code:

```
MoviePlayer1.movie=MyMovie
```

Using object binding, you can add pushbuttons that play or stop the movie. See the section on the **MoviePlayer** control in the *User's Guide*.

See Also OpenURLMovie function; Movie, EditableMovie classes.

MsgBox Method

Displays the string passed in an Alert box.

Syntax

MsgBox *message*

Part	Type	Description
message	String	Any valid string expression.

Notes

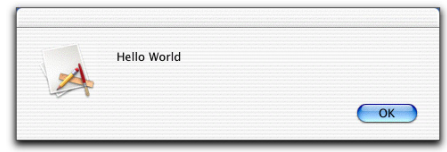
The Alert box opened by **MsgBox** has a Title bar. On Windows, the dialog also has a Close widget in its Title bar. The dialog can be closed either by clicking OK or clicking the Close widget. On Macintosh, the Alert has a fixed width; on Windows, the width increases to accommodate the longest paragraph. On Macintosh, text in a paragraph word-wraps to fit the width of the MsgBox.

Multiple paragraphs can be passed in the *message* parameter by separating each paragraph with the EndOfLine function.

Examples

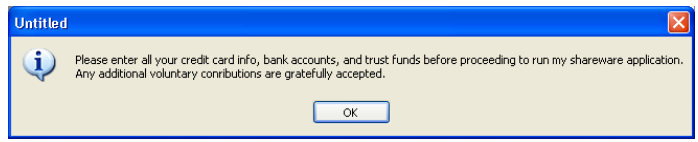
This example displays a one line Alert box.

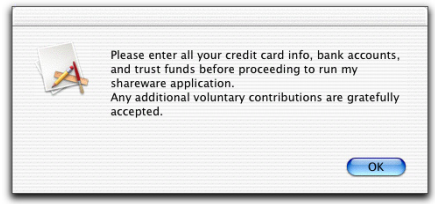
```
MsgBox "Hello World."
```



This example displays a multiline Alert box. Please note that it is entered as one line in the Code Editor.

```
MsgBox "Please enter all your credit card info, bank accounts, and trust funds before proceeding to run my shareware application."+EndOfLine+"Any additional voluntary contributions are gratefully accepted."+EndOfLine
```





See Also

EndOfLine function.

MySQLDatabase Class

Used to open MySQL databases.

Properties

Name	Type	Description
Port	Integer	The port that the MySQL database will use to connect. The default value is 3306.
Timeout	Integer	The connection timeout value (seconds). The default is 15.

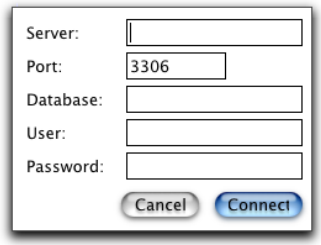
Methods

Name	Parameters	Description
GetAffectedRows		Returns as an Integer the number of rows that were modified by the most recent SQLExecute statement.
GetInsertID		Returns as an Integer the value of the AUTOINCREMENT field for the last row that was inserted.
LogPackets	Location as FolderItem	A debugging function that logs all network packets between the driver and server to the file specified by *Location*.

Notes

In order to use this class, you must install the MySQL database plug-in in your plugins folder.

If the plug-in is installed, you can also open a MySQL database in the REALbasic IDE using the File ▶ Add Data Source ▶ SelectMySQLDatabase command. This command presents a dialog box that requests the server, port, database, Username, and Password properties.



The set of database plug-ins is included on the REALbasic CD, but you may find more recent versions at the REAL Software web site, www.realsoftware.com.

See Also

Database class.

New Operator

Used to create new instances of objects at runtime.

Syntax

The **New** operator has two different syntaxes:

When creating new instances of controls:

object reference* = New *object name

Part	Type	Description
object reference	Any object type	Required; variable of type object name.
object name	Object name	Required. Any name of an object in the current window.

When creating new instances all other types of objects:

object reference* = New *object class

Part	Type	Description
object reference	Any object type	Required; variable of type object class.
object class	Object name	Required. Name of any object class that except the control classes.

Notes

Creating new instances of controls or menu items

The **New** operator can only create a new instance of a control that already exists in the window or a menu item that already exists on a menu. The object name acts as a template for creating the new instance of the control or menu item. The **New** operator works best if the control or menu item is in a control array — since otherwise you'll have trouble telling the old control from the new control. The object reference variable must be defined as type object name.

Creating new instances of all other types of objects

The **New** operator can be used to create a new instance of any type of object (for controls see the notes above). The object reference must be defined as being of type object class.

If the class's Visible property is set to True, the new instance will appear visibly in the running application.

Examples

This example creates a new instance of a pushbutton called "Pushbutton1" then sets its caption to "OK".

```
Dim pb As PushButton
pb=New pushbutton1
pb.caption="OK"
```

This example creates and then displays a new instance of a window called "AboutBox".

```
Dim w As aboutbox
w=New aboutbox
```

This example creates a new instance of a menu item called "WindowItem1".

```
Dim m As MenuItem
m=New WindowItem1
```

See Also

"Sharing Code Among An Array of Controls" on page 209 of the *User's Guide*; Nil, NewPicture functions.

NewAppleEvent Function

Creates a new AppleEvent class object.

Syntax

result*=NewAppleEvent*(eventClass, eventID, creatorCode)

Part	Type	Description
Result	AppleEvent Object	The new AppleEvent object.
EventClass	String	The four letter AppleEvent class code.
EventID	String	The four letter AppleEvent ID.
creatorCode	String	The four letter creator of the application the AppleEvent will be sent to.

Notes

Use this function to create a new AppleEvent to send data to another application. The EventClass and EventID together uniquely define a particular AppleEvent. The EventClass acts as a category for logically grouping events together. While there are many standard (and even some required) AppleEvents, many applications have several custom AppleEvents for performing actions specific to the application. Consult the application's documentation or its author to get information on what custom AppleEvents may be available. If you pass a blank string for the CreatorCode, the event will be "sent to self."

Once you create the AppleEvent object and populate the necessary parameters with data, you then send the AppleEvent to the target application using the AppleEvent object's Send method.

Examples

See the AppleEvent Class for an example of creating and sending an AppleEvent.

See Also

AppleEvent class.

NewAppleEventTarget Function

Used to specify the target application that will receive an Apple Event.

Syntax

***result*=NewAppleEventTarget(*Name*, *Computer*, *Zone*, *PortType*)**

Part	Type	Description
Result	AppleEventTarget	Object representing Apple Event Target
Name	String	Name of target application.
Computer	String	Computer on which the app is installed.
Zone	String	Appletalk zone
PortType	String	Four character Creator code identifying the target application.

Example

This example targets the Mailsmith application running on the Daystar computer.

```
Dim apptarget as AppleEventTarget
apptarget=NewAppleEventTarget("Mailsmith 1.1","Daystar","*","BLTO")
```

See Also

AppleEvent class.

NewDragItem Function

Defines the drag rectangle (based on the coordinates passed) that will appear when the user drags from the control or window. Returns a new DragItem object.

Syntax

***result*=NewDragItem(*Left*, *Top*, *Width*, *Height*)**

Part	Type	Description
Result	DragItem	Object representing the item being dragged.
Left	Integer	Left coordinate of the drag rectangle.
Top	Integer	Top coordinate of the drag rectangle.

Part	Type	Description
Width	Integer	Width of the drag rectangle
Height	Integer	Height of the drag rectangle.

Notes

You use **NewDragItem** in the an event handler of the control from which the user will drag. The MouseDown event handler is an obvious place. Your code uses **NewDragItem** to create the DragItem and size the drag region. Depending on the type of data being dragged (you can allow for multiple dragitems), you would use the FolderItem, Picture, MacData, or Text properties of the DragItem to store the item(s) being dragged.

If you are dragging from a RectControl, you can also use the NewDragItem method of the RectControl class to create a DragItem.

Example

The following code in the MouseDown event handler of an ImageWell creates a DragItem and enables dragging:

```
Dim d as DragItem
d=NewDragItem(Me.left,Me.top,Me.width,Me.height)
d.Picture=Me.image
d.Drag //Allow the drag
```

The following code in the Open event of another ImageWell enables it to receive a dragged picture:

```
Me.AcceptPictureDrop
```

The following DropObject event handler assigns the image contained in the DragItem to the control's BackDrop property. This causes it to be displayed.

```
Sub DropObject(obj as DragItem)
 Me.image=obj.Picture
End Sub
```

See Also

DragItem class.

NewMemoryBlock Function

Returns a new MemoryBlock class object of the size specified. Used for dealing with raw memory, and can be used with Declared functions and pointer parameters of PowerPC shared library entry points.

Syntax

***result*=NewMemoryBlock(*size*)**

Part	Type	Description
result	MemoryBlock	A new memory block based on the *size* specified.
size	Integer	The size (in bytes) of the new memory block.

Notes

Use the **NewMemoryBlock** function to create an empty MemoryBlock class object that can be used to pass data to a PowerPC shared library entry point or return data from an entry point.

If the application doesn't have sufficient memory to allocate to the memory block, Nil is returned. Check for Nil before proceeding.

Example

The following example creates a 384 byte memoryblock, whose bytes are numbered 0 to 383:

```
Dim m as MemoryBlock
m=NewMemoryBlock(384)
If m = Nil then
 Beep
 MsgBox "Insufficient memory to continue."
else
//proceed normally
end if
```

See Also

MemoryBlock class; Nil operator.

NewPicture Function

Returns a picture object of the size and depth specified. Used for off-screen drawing and direct pixel manipulation using the RGBSurface property of the Picture class.

Syntax

***result*=NewPicture(*width*, *height*, *depth*)**

Part	Type	Description
result	Picture	A new picture based on the *width*, *height*, and *depth* specified.
width	Integer	The width (in pixels) of the new picture.
height	Integer	The height (in pixels) of the new picture.
depth	Integer	The color depth of the new picture. Acceptable values are 0, 1, 2, 4, 8, 16, 32. A depth of zero indicates that the picture will be created with no pixel map at all; this option is for creating vector graphics, for example with the subclasses of the Object2D class.

Notes

Use the **NewPicture** function to create an empty picture object that can be used to create graphics that you don't want drawn to screen immediately. This is commonly referred to as "drawing to an off-screen area."

Pictures that are created with **NewPicture** with a color depth of 16 or 32 can be manipulated at the pixel level using the RGBSurface property of the Picture class. This is faster than equivalent operations using the Graphics class's methods.

If the application doesn't have enough memory to create the new picture, it will be Nil. The way to detect this is to immediately test for a Nil Picture object immediately after calling **NewPicture**. This is done in the example in the following section.

Alternatively, you can create a new picture using the New operator instead of using **NewPicture**. The parameters are the same in either case (width, height, depth), but when you use New, an OutOfMemoryException will be raised if there is insufficient memory to create the requested pixel map.

Example

The following example creates an animation in an off-screen area and then displays it in a Canvas control once the animation is complete. This approach produces a flicker-free animation.

```
Dim x,y as Integer
Dim buffer as Picture
'Build a buffer to hold the graphics while they
'are being assembled. Use the canvas Width and
'Height and the color depth on the main screen
Buffer=NewPicture(cvsImage.Graphics.Width,cvsImage.Graphics.Height,
Screen(0).Depth)
If Buffer <> Nil then
'Set the starting x and a random y
'position for the spaceship
 x=0
 y=Rnd*(cvsImage.Graphics.width-50)

'Loop until the x position of the ship is greater
'than the width of the canvas.
 Do
'Draw the background into the buffer
  buffer.Graphics.DrawPicture Space,0,0
'Draw the spaceship into the buffer
  buffer.Graphics.DrawPicture Ship,x,y
'Increment the ship's position
'Note: I increased the increment because this method takes a little longer.
   x=x+2
'Update the canvas with the buffer image
  cvsImage.Graphics.DrawPicture buffer,0,0
 loop until x>cvsImage.Graphics.Width
else 'buffer is Nil
 MsgBox "Insufficient memory to continue"
end if
```

The following code creates the animation directly in the Canvas control, resulting in flicker as the object is moved:

```
Dim x,y as Integer
'Set the starting x and a random y
'position for the spaceship
x=0
y=Rnd*(cvsImage.Graphics.width-50)

'Loop until the x position of the ship is greater
'than the width of the canvas.
Do
'Draw the background
 cvsImage.Graphics.DrawPicture Space,0,0
'Draw the spaceship
 cvsImage.Graphics.DrawPicture Ship,x,y
'Increment the ship's position
 x=x+1
loop until x>cvsImage.Graphics.Width
```

Whenever possible, use **NewPicture** to create smoother animations.

See Also Picture, Graphics classes, Canvas control; Nil operator.

NewREALDatabase Function

Creates a new REAL database. Please use the CreateREALdatabase method of the REALdatabase class to create a REALdatabase.

Syntax ***result*=NewREALDatabase(*File*)**

Part	Type	Description
result	Database	Reference to internal REAL database.
File	FolderItem	Reference to FolderItem that stores the database on disk.

Note

A REAL database is single-user and is limited to 8,174 bytes per record, 8,174 bytes per varchar field, and 254 columns per table. There is no fixed maximum number of rows.

Database-related features are fully supported in the Professional version of REALbasic. In the Standard version, you can experiment with all database features, but you are limited to 50 rows of data and you cannot build runtime database applications.

See the chapter, "Creating databases with REALbasic" in the *User's Guide* for information about the capabilities of the REAL database.

Example

The following example creates a new REAL database.

```
Dim dbFile as FolderItem
Dim db as Database
dbFile = GetFolderItem("Customers")
db = NewREALDatabase(dbFile)
```

See Also

Database class; Open4DDatabasebyADSP, Open4DDatabasebyTCPIP, OpenCSVCursor, OpenDBFCursor, OpenDTFDatabase, OpenODBCDatabase, OpenOracleDatabase, OpenOpenBaseDatabase, OpenREALDatabase, OpenPostgreSQLDatabase, Select4DDatabaseByADSP, SelectODBCDatabase, REALdatabase functions.

Nil Function

Used to determine if an object is nil (no value).

Syntax

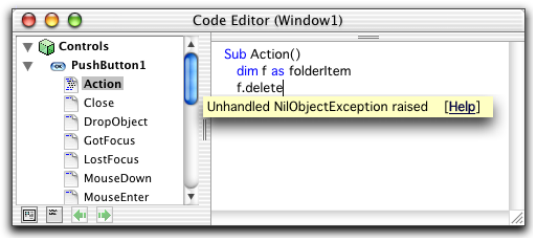
***expression*=Nil**

Notes

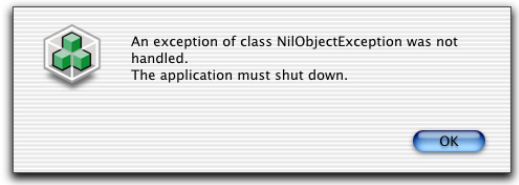
Nil means no value. Some functions will return a **Nil** value under certain circumstances. GetOpenFolderItem will return **Nil** if the user clicks the Cancel button when GetOpenFolderItem presents the standard open file dialog box.

Functions such as TrashFolder and ControlPanelsFolder return **Nil** if executed under Windows.

If you try to access an object that has a **Nil** value, a NilObjectException error will be generated. If you permit the error to be generated, execution will stop. If you are testing the application in the IDE, an error message will appear in your Code Editor:



If the code executes in a built application, the user will see a generic error message before the application quits.



To prevent this, you should always test for **Nil** values, either with an If statement or by including an Exception block at the end of the method.

Example

This examples uses the **Nil** function to determine if the user clicked the Cancel button in the standard open file dialog box presented by the GetOpenFolderItem function.

```
Dim f as FolderItem
f=GetOpenFolderItem("image/x-pict")
If f <> Nil Then
 //the user clicked OK so continue here
End If
```

See Also

RuntimeException class, Exception block, NilObjectException error.

NilObjectException Runtime Error

Occurs when code tries to access an object that doesn't exist.

Super Class

RuntimeExeception

Notes

NilObjectExceptions occur when you use a property or variable to access an object that doesn't exist. For example, when the GetFolderItem function is passed an invalid pathname, it returns Nil. Certain functions that are designed to return FolderItems to directories that exist on Macintosh operating systems return Nil when run on Windows. TrashFolder and ControlPanelsFolder, for example, return Nil under Windows. If your code attempts to access or write to a property of a Nil FolderItem, a **NilObjectException** will occur.

A function also returns Nil if there isn't enough memory to load the object. For example, if you tried to load a large picture into memory but your application didn't have enough RAM left to load the picture, the function used to load the picture would return Nil.

Immediately after calling a function that returns an object (like GetFolderItem, for example) you should check the object to see if it is Nil or add an Exception block to the end of the method that handles NilObjectExceptions. Any function's result that is not a String, Integer, Single, Double, Color, or Boolean is an object and should be checked.

Examples

This example tries to open a PICT file using GetFolderItem and display it in an ImageWell. If the pathname passed to GetFolderItem is invalid, it returns Nil. The **NilObjectException** is handled in the Exception block which is after the last "regular" line of the method.

If the pathname is valid, but the item does not exist, GetFolderItem returns a valid FolderItem whose AbsolutePath property is equal to the pathname. You should check the Exists property of the FolderItem to determine whether the file exists before trying to do something with the FolderItem.

This code, in other words, checks for two different problems—invalid pathname and valid path to a nonexistent file.

```
Sub Action ()
 Dim f as FolderItem
 f= GetFolderItem("HD:Images:Logo")
 If Not f.exists then
  Beep
  MsgBox "The file "+f.absolutePath+"doesn't exist!"
 else //document exists
  If f.MacType="PICT" then
  ImageWell1.image=f.OpenAsPicture
  end if
 end if
Exception err as NilObjectException
 MsgBox "Invalid pathname!"
End Sub
```

The following example creates a new, empty picture. If there isn't enough memory to create the picture, Nil will be returned by the NewPicture function. The example code checks for this and informs the user:

```
Dim p as Picture
p = NewPicture(100,100,32)
If p = Nil then
 Beep
 MsgBox "There isn't enough memory available to create the picture."
else
  //do something with the picture here
end if
```

See Also

RuntimeException class, OutofBoundsException error, IllegalCastException error, StackOverFlowException error, TypeMismatchException error, Function statement, Raise statement, Nil function, Exception block.

Not Operator

Used to perform logical negation of a Boolean expression.

Syntax

result*=Not *expression

Part	Description
result	A boolean value.
expression	Any valid boolean expression.

Notes

Not returns the opposite of the Boolean value *expression* evaluates to. If *expression* evaluates to True, **Not** returns False. If *expression* evaluates to False, **Not** returns True.

See Also

And, Or operators.

NotePlayer Control

Plays musical notes via the QuickTime Musical Instruments plug-in.

Super Class

Control

Because this is a Control, see the Control Class for other properties and events that are common to all Control objects.

Properties

Name	Type	Description
Index	Integer	The number of the control when it's part of a control array.
Instrument	Integer	The number of the musical instrument to be used to play the note. (see chart below)
Left	Integer	The left side of the control in local coordinates (relative to the window).
Name	String	The name of the object.
Top	Integer	The top of the control in local coordinates (relative to the window).

Methods

Name	Parameters	Description
PlayNote	Pitch as Integer, Velocity as Integer	Plays the note specified by the pitch at the velocity specified.

Notes

When calling the PlayNote method, the pitch is represented as a number from 0 to 127 where middle C is 60. Incrementing by 1 raises the pitch by a half step. Decreasing by 1 decreases the pitch by a half step. You can also express the value of pitch as a number from 256 to 32767, i.e. pitch*256. Use this to specify half-steps. For example, if you want a pitch of 60.5, specify 60.5*256=15488 as the value of Pitch.

The velocity parameter is a scale from 0 to 127 that represents how hard the key is pressed. 127 represents a very hard key press. 0 represents the key no longer being pressed. When calling the PlayNote method, the note played will continue to play until it is played again with a velocity of 0.

Instruments

Instrument	Value	Instrument	Value
Acoustic Grand Piano	1	Soprano Sax	65
Bright Acoustic Piano	2	Alto Sax	66
Electric Grand Piano	3	Tenor Sax	67
Honky-tonk Piano	4	Baritone Sax	68
Rhodes Piano	5	Oboe	69
Chorused Piano	6	English Horn	70
Harpsichord	7	Bassoon	71
Clavinet	8	Clarinet	72
Celesta	9	Piccolo	73
Glockenspiel	10	Flute	74
Music Box	11	Recorder	75
Vibraphone	12	Pan Flute	76
Marimba	13	Bottle blow	77
Xylophone	14	Shakuhachi	78
Tubular bells	15	Whistle	79
Dulcimer	16	Ocarina	80
Draw Organ	17	Square Lead	81
Percussive Organ	18	Saw Lead	82
Rock Organ	19	Calliope	83
Church Organ	20	Chiffer	84
Reed Organ	21	Synth Lead 5	85

Instrument	Value	Instrument	Value
Accordion	22	Synth Lead 6	86
Harmonica	23	Synth Lead 7	87
Tango Accordion	24	Synth Lead 8	88
Acoustic Nylon Guitar	25	Synth Pad 1	89
Acoustic Steel Guitar	26	Synth Pad 2	90
Electric Jazz Guitar	27	Synth Pad 3	91
Electric clean Guitar	28	Synth Pad 4	92
Electric Guitar muted	29	Synth Pad 5	93
Overdriven Guitar	30	Synth Pad 6	94
Distortion Guitar	31	Synth Pad 7	95
Guitar Harmonics	32	Synth Pad 8	96
Wood Bass	33	Ice Rain	97
Electric Bass Fingered	34	Soundtracks	98
Electric Bass Picked	35	Crystal	99
Fretless Bass	36	Atmosphere	100
Slap Bass 1	37	Bright	101
Slap Bass 2	38	Goblin	102
Synth Bass 1	39	Echoes	103
Synth Bass 2	40	Space	104
Violin	41	Sitar	105
Viola	42	Banjo	106
Cello	43	Shamisen	107
Contrabass	44	Koto	108
Tremolo Strings	45	Kalimba	109
Pizzicato Strings	46	Bagpipe	110
Orchestral Harp	47	Fiddle	111
Timpani	48	Shanai	112
Acoustic String Ensemble 1	49	Tinkle bell	113
Acoustic String Ensemble 2	50	Agogo	114
Synth Strings 1	51	Steel Drums	115
Synth Strings 2	52	Woodblock	116
Aah Choir	53	Taiko Drum	117
Ooh Choir	54	Melodic Tom	118
Synvox	55	Synth Tom	119
Orchestra Hit	56	Reverse Cymbal	120
Trumpet	57	Guitar Fret Noise	121

Instrument	Value	Instrument	Value
Trombone	58	Breath Noise	122
Tuba	59	Seashore	123
Muted Trumpet	60	Bird Tweet	124
French Horn	61	Telephone Ring	125
Brass Section	62	Helicopter	126
Synth Brass 1	63	Applause	127
Synth Brass 2	64	Gunshot Table	128
		Drum Kit	16385

Examples

The following plays a note on the 'Church Organ' instrument:

```
NotePlayer1.Instrument=20
NotePlayer1.PlayNote(60,60)
```

The following line of code stops the note:

```
NotePlayer1.PlayNote(60,0)
```

See Also

Control class.

NthField Function

Returns a field from a row of data. The first field is numbered 1.

Syntax

***result*=NthField(*source*, *separator*, *fieldNumber*)**

OR

***result=stringVariable*.NthField(*separator*, *fieldNumber*)**

Part	Type	Description
result	String	The field value desired.
source	String	The fields of data.
separator	String	The character that separates the columns of data.
fieldNumber	Integer	The column number of the field of data desired.
stringVariable	String	Any variable of type String.

Notes

The **NthField** function returns the field value from the *source* that precedes the *fieldNumber* occurrence of the *separator* in the *source*.

If *fieldNumber* is out of bounds, an empty string is returned. **NthField** is not case-sensitive.

Examples

This example returns "Smith".

```
Dim field As String
field=NthField("Dan*Smith*11/22/69*5125554323*Male","*",2)
```

Using the second syntax:

```
Dim s,field As String
s="Dan*Smith*11/22/69*5125554323*Male"
field=s.NthField("*",2)
MsgBox field
```

See also the example that illustrates how to populate a PopupMenu control.

See Also

TextInputStream example, CountFields function.

Object Class

Object is the base class of all other classes. Any object can be assigned into a variable of type **Object**. The object class has no properties, methods, or events.

Object2D Class

The base class of any object that can be drawn in a vector graphics environment. REALbasic 4.5 and above supports vector graphics.

Super Class

Object

Properties

Name	Type	Description
Border	Double	Opacity of the border, from 0 (transparent) to 100 (opaque). The default is 0.
BorderColor	Color	Color of the border.
BorderWidth	Double	Width of the border, in points. The default is 1.
Fill	Double	Opacity of the interior, from 0 (transparent) to 100 (opaque). The default is 100.
FillColor	Color	Color of the interior of the shape.

Name	Type	Description
Rotation	Double	Clockwise rotation, in radians, around the X, Y point.
Scale	Double	Scaling factor relative to the object's original size
X	Integer	Horizontal position of center or main anchor point.
Y	Integer	Vertical position (down from top) position of center or anchor point.

Notes

Object2D is the base class for a group of subclasses that enable you to create vector (as opposed to bitmap) graphics. The subclasses are shown in the following table:

Class	Description
ArcShape	Draws an arc of a circle.
CurveShape	Draws straight lines or curves using one or more "control points."
FigureShape	Draws polygons that can (optionally) have curved sides.
OvalShape	Draws circles and ovals.
PixMapShape	Imports a bitmap picture into the image.
RectShape	Draws a square or rectangle.
RoundRectShape	Draws a square or rectangle with rounded corners (subclassed from RectShape).
StringShape	Draws text strings in a specified font, font size, and style.

These basic shapes can be organized into a hierarchy using the Group2D class.

The Picture class has a new property, Objects, that enables you to include a vector graphics drawing (a Group2D object) in a picture and the Graphics class has a new method, DrawObject, that enables you to draw a Group2D object within the Graphics object. Also, the SaveAsPicture method of the FolderItem class now has an optional parameter that enables you to save a vector graphics picture in a format that retains the vector information. The OpenAsVectorPicture method of the FolderItem class attempts to open an existing image as a vector graphic and map the contents of the image into Object2D drawing primitives.

Examples

See the examples for ArcShape, CurveShape, FigureShape, OvalShape, PixmapShape, RectShape, RoundRectShape, and StringShape.

See Also

ArcShape, CurveShape, FigureShape, FolderItem, Group2D, Graphics, OvalShape, Picture, PixmapShape, RectShape, RoundRectShape, StringShape classes.

Object3D Class

The basic 3D object type.

Properties

Name	Type	Description
Bounds	Bounds3D	Stores the bounding volume for the **Object3D**. The default value is Nil. If you assign a Bounds3D here, it will be automatically updated when the object is rendered or when you call UpdateBounds.
NullShader	Boolean	If set to True, causes the object to be rendered without respect to external lighting, as if you had set AmbientLight=100 and FloodLight=0). Useful for objects that should appear to glow from within, such as torches and explosions, or for backdrops or sprites that have lighting pre-calculated as part of the texture.
Orientation	Quaternion	Angle of the object in 3D space. Can also be controlled by the Pitch, Roll, and Yaw methods.
Position	Vector3D	The position (i.e., x, y, z coordinates) of the object in 3D space.
RenderBack Faces	Boolean	Allows you to decide on an object-by-object basis whether polygons facing away from the camera should be drawn. The default value is False, meaning such polygons should be skipped for faster rendering.
Scale	Double	Determines the size of the model in 3D space; a scale of 1.0 means that it is drawn at the same size it was when it was created.
Shape	Integer	Selects which of several shapes is currently being drawn. The first shape added is numbered 0. Select this shape by setting Shape=0, and so forth.
Visible	Boolean	True if visible.

Methods

Name	Parameters	Description
AddShapeFromFile	File as FolderItem	Loads a model file and adds it to an object in one step. If the file does not exist or does not contain valid model data, then no shape is added.
AddShapeFromHandle	data as Integer	Adds a shape from geometry which has been created through Declare statements.
AddShapeFromString	data as String	Loads a model from 3DMF data in a string.

Name	Parameters	Description
AddShapePicture	image as Picture, scale as Double	Creates a flat, rectangular model from a REALbasic picture, with white pixels appearing transparent. A scale of 1.0 means that each pixel of the picture is 1 unit across in 3D space.
AddShapePictureWithMask	image as Picture, mask as Picture scale as Double	LIke AddShapePicture, but specifies a second picture as a mask to create a translucent image.
Clone		Clones the object and returns another object of the same class as the original, even when the original is a subclass of **Object3D**. The clone has the same geometry as the original. Returns an **Object3D**.
ComputeBounds	Type as Integer	Returns a Bounds3D bounding volume of type *Type*. Typically used to create and assign the object's bounding volume to the Bound property,
GetShapeHandle	Index as Integer	Returns (as an Integer) the TQ3GeometryObject of the specified shape.
MoveForward	distance as Double	Moves the object in whatever direction it is facing.
ObjectToWorldTransform	pt as Vector3D	Returns a Vector3D. Takes a vector specifying a point relative to the object, and returns the coordinates of that point in the 3D world, taking into account the object's current scale, orientation, and position.
Pitch	radians as Double	Rotates the object in the vertical plane.
Roll	radians as Double	Rotates the object clockwise or counterclockwise.
UpdateBounds		Updates the Bounds3D object attached to the object via its Bounds property. In the case of a Group3D, this also updates the bounds of any objects contained in the group. This is done automatically to the Rb3DSpace.objects group when you call Rb3DSpace.Update.
WorldToObjectTransform	pt as Vector3D	Takes a vector specifying a point in the 3D world, and returns the coordinates of that point relative to the object, taking into account the object's current scale, orientation, and position. Returns a Vector3D.
Yaw	radians as Double	Rotates the object in the horizontal plane.

Notes

This is the basic three-dimensional object type. An **Object3D** has a position, an orientation, and it may have multiple geometry models. Use AddShapeFromString to load a model from 3DMF data in a string, or use AddShapePicture to create a flat, rectangular model from a REALbasic picture, with white pixels appearing transparent. The scale parameter determines the size of the model in 3D space; a scale of 1.0 means that each pixel in the picture is 1 unit across in 3D space. You can change its position and orientation by directly modifying its properties, or you can use the Roll, Pitch, and Yaw methods to rotate the object, and MoveForward to move it in whatever direction it's facing (considered to be the +Z direction initially).

You can change its scale by modifying the Scale property; this defaults to 1.0. If you need more than one copy of an object in the game (as is often the case), use the Clone method which returns a new object that shares the same geometry as the original. Because the geometry is shared, this is much more economical than loading separate, duplicate copies of the model.

See Also

RB3DSpace control, Vector3D, Group3D, Quaternion, Bounds3D, Light3D classes.

OLEContainer Control

The generic container for embedding ActiveX controls.

Super Class

RectControl

Properties

Name	Type	Description
Content	OLEObject	The OLEObject associated with the ActiveX control. You use this to automate the ActiveX control.
ProgramID	String	The ActiveX control's program ID as stored in the registry. It can also be the GUID (include curly braces).

Methods

Name	Parameters	Description
Create		Creates an ActiveX control. Create uses the ProgramID property to create the ActiveX control. Returns a Boolean: True if successful, False if unsuccessful.
Destroy		Destroys the currently embedded ActiveX control, so you can reuse your OLEContainer.

Name	Parameters	Description
ShowPropertyPages		Displays the ActiveX control's Property pages, if any. Not all ActiveX controls support Property pages.

Events

Name	Parameters	Description
EventTriggered	NameOfEvent as String Parameters() as array of Variants	Occurs when the ActiveX control receives an event from the user. The event name is passed as the first parameter specifying which event occurred. The second parameter contains the parameters for this event, passed in as an array of variants. The user should check the UBound of the parameters list.
ShowObject		Occurs when the ActiveX object is ready to be displayed.

See Also

OLEObject class.

OLEException Error

An OLE-related runtime error.

Super Class

RuntimeException

OLEObject Class

Used to automate COM servers.

SuperClass

Object

Constructors

Name	Parameters	Description
OLEObject	ProgramID as String	*ProgramID* is the COM server's program ID as stored in the registry. It can also be the Class ID (in curly braces). This constructor will try to find a previous instance of the COM server if it's running. Otherwise, it will create a new instance.
OLEObject	ProgramID as String NewInstance as Boolean	The ProgramID parameter is the same as above. The second parameter specifies whether to create a new instance of the COM server (True) or try to use an existing one if it is running (False).

Methods

Name	Parameters	Description
Invoke	NameOfFunction as String	Invokes a method of the COM server. Returns a Variant.
Invoke	NameOfFunction as String Parameters() as array of Variants	Invokes a method of the COM server, and passes the array of parameters to the method. Make sure to correctly dimension the array, as this will determine the number of parameters that get passed to the method. The first parameter begins at 1.
Property	PropertyName as String	Used to get or set a property of the object. The parameter is the name of the property to assign a new value to.

Example

The following example automates Internet Explorer.

```
Dim obj as OLEObject
Dim v as Variant
Dim params(1) as Variant

// InternetExplorer.Application is the program ID,
// the class ID is {0002DF01-0000-0000-C000-000000000046}
// RB will accept either format
//obj = New OLEObject("{0002DF01-0000-0000-C000-000000000046}", True)
obj = New OLEObject("InternetExplorer.Application", True)
obj.property("Visible") = True
params(1) = "http://www.realsoftware.com/"
//v = obj.invoke("Navigate", params)
//v = obj.invoke("this_will_throw_an_exception")

Exception err as OLEException
  Msgbox err.message
```

See Also

OLEContainer class; OLEException error.

ODBCDatabase Class

Used to open an ODBC database. Use the **ODBCDatabase** class instead of OpenODBCDatabase.

Super Class

Database

Properties

Name	Type	Description
Attribute	Integer	Optional attribute. See docs on ODBC for SQLSetConnectAttr.
AttributeString	String	Optional attribute. See docs on ODBC for SQLSetConnectAttr.
DataSource	String	The connection string to be used to establish a connection to an ODBC database.

Notes

In order to use this class, you must install the appropriate database plug-in in your plugins folder.

The set of database plug-ins is included on the REALbasic CD, but you may find more recent versions at the REAL Software web site, www.realsoftware.com.

See Also

Database class, OpenODBCDatabase function.

Oct Function

Returns as a string, the octal version of the number passed.

Syntax

***result*=Oct(*value*)**

Part	Type	Description
result	String	The value passed converted to octal.
value	Integer	The number to be converted to octal.

Notes

If the *value* is not a whole number, the decimal value will be truncated.

VB Compatibility Note: VB rounds the value to the nearest whole number so the **Oct** function will probably be changed in a future release to do this as well.

You can specify binary, hex, or octal numbers by preceding the number with the & operator and the letter that indicates the number base. The letter b indicates binary, h indicates hex, and o indicates octal.

Examples Below are examples of various numbers converted to octal:

```
Dim OctVersion As String
OctVersion=Oct(5)    //returns "5"
OctVersion=Oct(75)   //returns "113"
OctVersion=Oct(256)  //returns "400"
```

See Also & operator, Bin and Hex functions.

One of the Interfaces of this Class is not of Type Class Interface Error

In the Properties Window for a Class Interface, you entered the name of a class in the Interfaces field that is not of type Class Interface. This occurs if you use the name of any 'ordinary' class instead of the name of a Class Interface.

Only Boolean Constants can be Used with #If Error

Only the Boolean constants TargetMacOS, TargetPPC, TargetCarbon, TargetWin32, DebugBuild, and Target68K (obsolete) can be used in the #if statement.

Example A 'regular' boolean variable cannot be used as shown here.

```
Dim b as Boolean
b=True
#if b then
 Beep
#endif
```

See Also TargetMacOS, TargetPPC, TargetCarbon, TargetWin32, DebugBuild, Target68K constants; #If statement.

Only String Constants Can Be Used For Declaring Libraries Error

Occurs when you've declared an external function using a constant, but either the constant is not defined or it contains something other than a string.

Open4DDatabasebyADSP Function

Opens a 4th Dimension (4D Server) database using ADSP. Please use the ADSP4DServer class instead.

Syntax

***result*=Open4DDatabasebyADSP(*connectionstring*, *task*, *username*, *password*)**

Part	Type	Description
result	Database	Database object referring to 4D Server database.
connectionstring	String	4D Connection string.
task	String	4D task.
username	String	Username of database.
password	String	Password.

Notes

Open4DDatabasebyADSP requires the 4D ADSP plug-in for the version of 4th Dimension that you are running (6.0, 6.5, or 6.7). Place the correct plug-in in the Plugins folder. The REALbasic CD includes all database plug-ins and you can always find the most current versions on REAL Software's web site, www.realsoftware.com.

To access 4D Server, it must be configured to allow 4D Open connections and the user *username* must be in the group that has 4D Open access.

You can obtain the value of *connectionstring* using the Property method of the Database class. It contains information about the name and location of the database on the network (i.e., the network zone). Once you have obtained this value, you to connect to 4D Server using **Open4DDatabasebyADSP**, bypassing the dialog box that is displayed by Select4DDatabaseByADSP.

The *task* parameter is the name of the process that is created on 4D Server when you establish your connection. The value of task is displayed in the 4D Server window.

Example

```
Dim db as Database
db=Open4DDatabasebyADSP("connectstring","RBtask","Melissa","Peanuts")
If db = Nil then
 Beep
 MsgBox "The database couldn't be opened."
else
  //continue with database operations
end if
```

See Also

Database, ADSP4DServer classes; Open4DDatabasebyTCPIP, OpenCSVCursor, OpenDBFCursor, OpenDTFDatabase, OpenODBCDatabase, OpenOracleDatabase,

OpenOpenBaseDatabase, OpenPostgreSQLDatabase, OpenREALDatabase, Select4DDatabaseByADSP, Select4DDatabaseByTCPIP, SelectODBCDatabase functions.

Open4DDatabasebyTCPIP Function

Opens a 4th Dimension (4D Server) database using TCP/IP. This function has been superseded by the TCPIP4DServer class.

Syntax

***result*=Open4DDatabasebyTCPIP(*IPAddress*, *task*, *username*, *password*)**

Part	Type	Description
result	Database	Database object referring to 4D Server database.
IPAddress	String	IP address of server machine.
task	String	4D task.
username	String	Username of database.
password	String	Password.

Note

Open4DDatabasebyTCPIP requires the 4D TCPIP plug-in for the version of 4th Dimension that you are running (e.g., 6.0 or 6.7). Place the correct plug-in in the Plugins folder.

The REALbasic CD includes all database plug-ins and you can always find the most current versions on REAL Software's web site, www.realsoftware.com.

To access 4D Server, it must be configured to allow 4D Open connections and the user *username* must be in the group that has 4D Open access.

Example

```
Dim db as Database
db=Open4DDatabasebyTCPIP("192.168.1.5","mytask","Melissa","Peanuts")
If db = Nil then
 Beep
 MsgBox "The database couldn't be opened."
else
  //continue with database operations
end if
```

See Also

Database, TCPIP4DServer classes; NewREALDatabase, Open4DDatabasebyADSP, OpenCSVCursor, OpenDBFCursor, OpenDTFDatabase, OpenODBCDatabase, OpenOracleDatabase, OpenOpenBaseDatabase, OpenPostgreSQLDatabase,

OpenREALDatabase, Select4DDatabaseByADSP, Select4DDatabaseByTCPIP, SelectODBCDatabase functions.

OpenBaseDatabase Class

Used to open an OpenBase database. Please use this class rather than OpenOpenBaseDatabase.

Super Class

Database

Properties

Name	Type	Description
Encoding	Integer	This encoding value is the same value you'd use for GetTextEncoding. For example, &H201 is ISOLatin1, &H202 is ISOLatin2, and so forth. This is used to convert from your specified encoding to the OpenBase database encoding, and vice versa. This is included for 4.0.2 compatibility and is deprecated in version 4.5 and above.

Notes

In order to use this class, you must install the appropriate database plug-in in your plugins folder.

The set of database plug-ins is included on the REALbasic CD, but you may find more recent versions at the REAL Software web site, www.realsoftware.com.

Example

The following code opens the OpenBase database, "pubs."

```
Dim obdb as OpenBaseDatabase
Obdb = New OpenBaseDatabase
Obdb.host = "opendb.mydomain.com"
Obdb.databasename = "pubs"
If obdb.connect() then
 MsgBox "good"
Else
 MsgBox "bad"
end if
```

See Also

Database class.

OpenCSVCursor Function

Opens a comma-delimited text file as a RecordSet. The file must use commas to separate fields, returns to separate records, and double quotes ("") to enclose text fields. Commas are permitted within quoted strings. The following record is valid:

22,"Fred","Duesenberg","Engineer"

Syntax

***result*=OpenCSVCursor(*file*)**

Part	Type	Description
result	RecordSet	RecordSet containing the rows and columns in the text file.
file	FolderItem	FolderItem that refers to the text file.

Note

OpenCSVCursor requires the CSV Database plug-in. Place this plug-in in the REALbasic Plugins folder. The REALbasic CD includes all database plug-ins and you can always find the most current versions on REAL Software's web site, www.realsoftware.com.

Example

The following example opens the specified text file.

```
Dim db as RecordSet
db=OpenCSVCursor(GetFolderItem("table1.txt"))
If db = Nil then
 Beep
 MsgBox "The database couldn't be opened."
else
  //continue with database operations
end if
```

See Also

Database, RecordSet, DatabaseField classes; NewREALDatabase, Open4DDatabasebyTCPIP, Open4DDatabasebyADSP, OpenDBFCursor, OpenDTFDatabase, OpenODBCDatabase, OpenOracleDatabase, OpenOpenBaseDatabase, OpenREALDatabase, Select4DDatabaseByADSP, Select4DDatabaseByTCPIP, SelectODBCDatabase functions.

OpenDialog Class

Creates a customizable Open-file dialog box. The non-customizable version of this dialog is displayed by the GetOpenFolderItem function.

Super Class

FolderItemDialog.

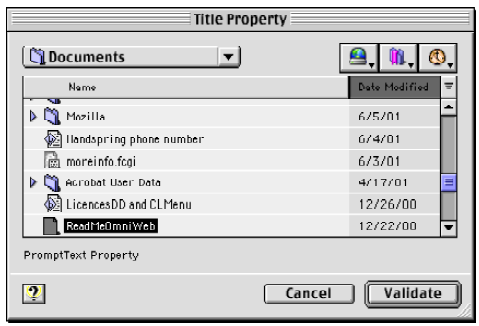
Notes

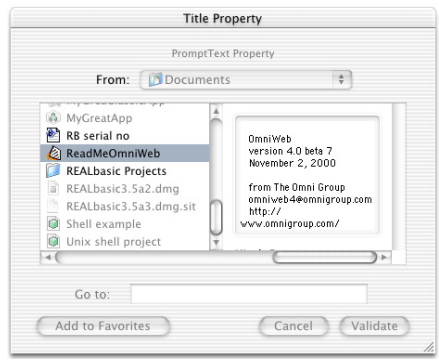
Using the properties of the FolderItemDialog class, you can customize the following aspects of an open-file dialog box:

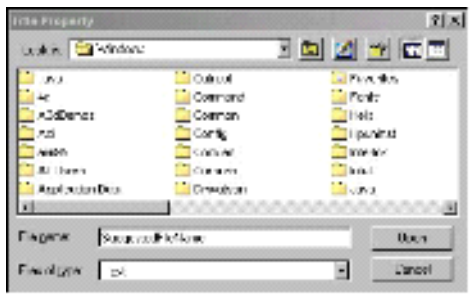
- Position (Left and Top properties)
- Default directory (Initial Directory property)
- Valid file types to show (Filter property)
- Text of the Validate and Cancel buttons (ActionButtonCaption and CancelButtonCaption properties)
- Text that appears in the Title bar of the dialog (Title property)
- Text that appears inside the dialog (PromptText property)

Example

This is what it looks like:







The following code opens a customizable open-file dialog box and presents the OnlineManuals directory in the file browser. Only MIF files are listed. The file type passed to the Filter parameter was previously defined in the File Types dialog box.

```
Dim dlg as OpenDialog
Dim f as FolderItem
dlg=New OpenDialog
dlg.InitialDirectory=GetFolderItem("HD:FrameMaker6.0:OnlineManuals")
dlg.Title="Select a MIF file"
dlg.Filter="application/x-mif"
f=dlg.ShowModal()
If f <> Nil then
 //Display full path to opened file in an EditField
 resultEdit.text = dlg.Result.AbsolutePath
Else
 resultEdit.text = "User Cancelled"
End if
```

See Also

FolderItemDialog, SelectFolderDialog, SaveAsDialog, FolderItem classes; GetOpenFolderItem function.

OpenDBFCursor Function

Opens an xBase format file (i.e., a dbf file from dBase) as a RecordSet.

Syntax

***result*=OpenDBFCursor(*file*)**

Part	Type	Description
result	RecordSet	RecordSet containing the rows and columns in the .dbf file.
file	FolderItem	FolderItem that refers to the .dbf file.

Example

```
Dim db as RecordSet
db=OpenDBFCursor(GetFolderItem("phone.dbf"))
If db = Nil then
 Beep
 MsgBox "The database couldn't be opened."
else
  //continue with database operations
end if
```

Note

OpenDBFCursor requires the DBF Database plug-in. Place this plug-in in the Plugins folder. The REALbasic CD includes all database plug-ins and you can always find the most current versions on REAL Software's web site, www.realsoftware.com.

See Also

Database, RecordSet classes; NewREALDatabase, Open4DDatabasebyTCPIP, Open4DDatabasebyADSP, OpenDTFDatabase, OpenCSVCursor, OpenODBCDatabase, OpenOracleDatabase, OpenPostgreSQLDatabase, OpenOpenBaseDatabase, OpenREALDatabase, Select4DDatabaseByADSP, Select4DDatabaseByTCPIP, SelectODBCDatabase functions.

OpenDTFDatabase Function

Opens a dtF database.

Syntax

***result*=OpenDTFDatabase (*dbFile*,*blobFile*,*username*,*password*)**

Part	Type	Description
result	Database	Database object that refers to the dtF database.
dbFile	FolderItem	All the fixed-length fields in the dtF file.
blobFile	FolderItem	All the blob fields in the dtF file.
username	String	Username for the dtF database.
password	String	Password for the dtF database.

Notes

OpenDTFDatabase requires the dtfSQL Database plug-in. Place this plug-in in the REALbasic Plugins folder. The REALbasic CD includes all database plug-ins and you can always find the most current versions on REAL Software's web site, www.realsoftware.com.

This function requires the Dtf_PPC shared library be installed in extensions folder

Example

```
Dim db as Database
Dim textdata as FolderItem
Dim blobdata as FolderItem
textdata=GetFolderItem("HD:mydb")
blobdata=GetFolderItem("HD:myblobs")
db=OpenDTFdatabase(textdata, blobdata,"Nancy","Pavlov")
If db = Nil then
  Beep
  MsgBox "The database couldn't be opened."
else
  //continue with database operations
end if
```

See Also

Database, RecordSet classes; NewREALDatabase, Open4DDatabasebyTCPIP, Open4DDatabasebyADSP, OpenDBFCursor, OpenODBCDatabase, OpenOracleDatabase, OpenOpenBaseDatabase, OpenPostgreSQLDatabase, OpenREALDatabase, Select4DDatabaseByADSP, Select4DDatabaseByTCPIP, SelectODBCDatabase functions.

OpenODBCDatabase Function

Opens an ODBC database without displaying an open-file dialog box, as does SelectODBCDatabase. This function has been superseded by the ODBCDatabase class.

Syntax

***result*=OpenODBCDatabase(*datasource*)**

Part	Type	Description
result	Database	Database object that refers to ODBC database.
datasource	String	ODBC database to be opened. May optionally contain the user ID and password; see Notes.

Notes

OpenODBCDatabase requires the ODBC Database plug-in. Place this plug-in in the REALbasic Plugins folder. The REALbasic CD includes all database plug-ins and you can always find the most current versions on REAL Software's web site, www.realsoftware.com.

SelectODBCDatabase also requires a third-party driver manager and driver. REALbasic supports OpenLink's ODBC Driver manager and drivers (www.openlinksw.com) and Merant/Metro Technologies ODBC Manager. OpenLink supports both Mac OS 8/9 and Mac OS X, while Merant currently does not support Mac OS X. A driver manager must be installed as well as the driver for whichever

database back end you wish to access. Check REAL Software's web site, www.realsoftware.com, for updated drivers.

The user ID and password can be passed using the following syntax:

```
db=OpenODBCDatabase("DSN=MyDataSource;UID=Ben;PWD=Jerry")
```

Example

```
Dim db as Database
db=OpenODBCDatabase("mydatabase")
If db = Nil then
 Beep
 MsgBox "The database couldn't be opened."
else
  //continue with database operations
end if
```

See Also

NewREALDatabase, Open4DDatabasebyTCPIP, Open4DDatabasebyADSP, OpenDBFCursor, OpenDTFDatabase, OpenOracleDatabase, OpenOpenBaseDatabase, Select4DDatabaseByADSP, Select4DDatabaseByTCPIP, SelectODBCDatabase functions; Database, RecordSet classes.

OpenOpenBaseDatabase Function

Opens an OpenBase database. This function has been superseded by the OpenBaseDatabase class instead.

Syntax

***result*=OpenOpenBaseDatabase (*host*, *database*, *username*, *password*)**

Part	Type	Description
result	Database	Database object that refers to OpenBase database
host	String	IP/DNS address
database	String	Name of the database on the OpenBase server.
username	String	Database username.
password	String	Database password.

Notes

OpenOpenBaseDatabase requires the Openbase plug-in. Place this plug-in in the REALbasic Plugins folder. The REALbasic CD includes all database plug-ins and you can always find the most current versions on REAL Software's web site, www.realsoftware.com.

If you use the RecordSet class to update a record using edit and update you need to include the "_rowid" column of the table. If you don't, OpenBase won't be able to find the record in order to update it.

Caution: Inserting data into Object fields in OpenBase can corrupt the database. The workaround for this is to insert the record without the object fields and then use the Update method to add the data for the object fields.

Example

```
Dim db as Database
db=OpenOpenBaseDatabase("192.168.7.1","MyDatabase","Marci","vertigo")
If db = Nil then
  Beep
  MsgBox "The database couldn't be opened."
else
  //continue with database operations
end if
```

See Also

Database, RecordSet OpenBaseDatabase classes; NewREALDatabase, Open4DDatabasebyADSP, Open4DDatabasebyTCPIP, OpenCSVCursor, OpenDBFCursor, OpenDTFDatabase, OpenODBCDatabase, OpenOracleDatabase, OpenPostgreSQLDatabase, OpenREALDatabase, Select4DDatabaseByADSP, Select4DDatabaseByTCPIP, SelectODBCDatabase functions.

OpenOracleDatabase Function

Opens an Oracle database.

Syntax

***result*=OpenOracleDatabase(*username*,*password*)**

Part	Type	Description
result	Database	Database object that refers to the Oracle database.
username	String	Oracle database username. The user name should also contain the name of the datasource.
password	String	Oracle database password

Notes

OpenOracleDatabase requires the Oracle (7.1) Database plug-in. Place this plug-in in the REALbasic Plugins folder. The REALbasic CD includes all database plug-ins and you can always find the most current versions on REAL Software's web site, www.realsoftware.com.

To use OpenOracleDatabase, OCI shared libraries must be installed in the Extensions folder. To access databases on remote machines, SQL*Net must also be installed.

Example

```
Dim db as Database
db=OpenOracleDatabase("Allison@nickelodeon","rugRat")
If db = Nil then
  Beep
  MsgBox "The database couldn't be opened."
else
  //continue with database operations
end if
```

See Also

Database, RecordSet classes; NewREALDatabase, Open4DDatabasebyTCPIP, Open4DDatabasebyADSP, OpenCSVCursor, OpenDBFCursor, OpenODBCDatabase, OpenPostgreSQLDatabase, OpenREALDatabase, OpenOpenBaseDatabase, Select4DDatabaseByADSP, Select4DDatabaseByTCPIP, SelectODBCDatabase functions.

OpenPostgreSQLDatabase Function

Opens a Postgre database. This function is superseded by the PostgreSQLDatabase class.

Syntax

***result*=OpenPostgreSQLDatabase(*host*,*port*,*database*,*user*,*password*)**

Part	Type	Description
result	Database	Database object that refers to the Postgre database.
host	String	Host computer
port	Integer	Port. PostgreSQL's default port is 5432.
database	String	PostgreSQL Database.
user	String	Username
password	String	Password

Notes

OpenPostgreSQLDatabase requires the PostgreSQL Database plug-in. Place this plug-in in the REALbasic Plugins folder. The REALbasic CD includes all database plug-ins and you can always find the most current versions on REAL Software's web site, www.realsoftware.com.

A field specified as type Float is actually implemented by postgreSQL as a double (8 bytes).

The IsPrimary column of the FieldSchema for a postgreSQL database always returns False.

PostgreSQL dates: field types Date, Time, and TimeStamp are now properly supported; use the DateColumn and DateValue methods for a date representation of this data, or use StringValue to get a human-readable date and/or time.

Example

```
Dim db as Database
db=OpenPostgreSQLDatabase("HostComp",5432,"MyDatabase","Marci","vertigo")
If db = Nil then
  Beep
  MsgBox "The database couldn't be opened."
else
  //continue with database operations
end if
```

See Also

Database, RecordSet, PostgreSQLDatabase classes; NewREALDatabase, OpenREALDatabase, Open4DDatabasebyADSP, Open4DDatabasebyTCPIP, OpenCSVCursor, OpenDBFCursor, OpenDTFDatabase, OpenODBCDatabase, OpenOracleDatabase, OpenOpenBaseDatabase, Select4DDatabaseByADSP, Select4DDatabaseByTCPIP, SelectODBCDatabase functions.

OpenPrinter Function

Returns a Graphics object in order to print to the currently selected printer.

Syntax

***result*=OpenPrinter(*[PageSetup]*)**

Part	Type	Description
result	Graphics	A graphics object that represents the page to be printed.
PageSetup	PrinterSetup	Optional. A PrinterSetup object whose properties will be used (like page orientation, scale, etc.) when printing.

Notes

The **OpenPrinter** function returns a graphics object. The various drawing routines can then be used to draw into the graphics object and will be sent to the printer for printing.

Examples

This example prints "Hello World" to the currently selected printer.

```
Dim g As Graphics
g  = OpenPrinter()
g.DrawString "Hello World", 100, 100
```

This example uses the Page Setup properties stored in the variable "ps" during printing:

```
Dim g As Graphics
g  = OpenPrinter(ps)
g.DrawString "Hello World", 100, 100
```

See Also

Graphics class, OpenPrinterDialog function, PrinterSetup class.

OpenPrinterDialog Function

Displays the standard Print dialog box and returns a Graphics object in order to print to the currently selected printer.

Syntax

***result*=OpenPrinterDialog(*[PageSetup],[window]*)**

Part	Type	Description
result	Graphics	A graphics object that represents the page to be printed.
PageSetup	PrinterSetup	Optional. A PrinterSetup object whose properties will be used (like page orientation, scale, etc.) when printing.
window	Window	Optional. A sheet window that you want to use as the Page Setup sheet window instead of the built-in Page Setup dialog.

Notes

The **OpenPrinterDialog** function returns a Graphics object. The various drawing routines can then be used to draw into the Graphics object and will be sent to the printer for printing. You can get the values in the Copies, From and To fields of the Print dialog box using the Copies, FirstPage and LastPage properties of the Graphics object returned by the **OpenPrinterDialog** function.

If the user clicks the Cancel button, the **OpenPrinterDialog** function returns Nil.

Since both parameters are optional, there is the possibility that you will want to pass the second parameter but not the first. If you pass only one parameter, REALbasic will assume that it is the first parameter. If you want to pass only the second parameter, pass Nil as the value of the first parameter, for example:

```
Dim g As Graphics
g = OpenPrinterDialog(Nil, myWindow)
```

Examples

This example prints "Hello World" to the currently selected printer.

```
Dim g As Graphics
g  = OpenPrinterDialog()
If g<> Nil then
  g.DrawString "Hello World", 100, 100
End if
```

This example uses the Page Setup properties stored in the variable "ps" during printing:

```
Dim g As Graphics
g  = OpenPrinterDialog(ps)
g.DrawString "Hello World", 100, 100
```

See Also

Graphics, PrinterSetup, StyledTextPrinter classes; OpenPrinter function.

OpenREALDatabase Function

Opens an existing REAL database. This function has been superseded by the REALDatabase class. Please use this class instead.

Syntax

***result*=OpenREALDatabase(*file*)**

Part	Type	Description
result	Database	Database object that refers to the REAL database.
file	FolderItem	REAL database to be opened.

Notes

A REAL database is single-user and is limited to 8,174 bytes per record, 8,174 bytes per varchar field, and 254 columns per table. There is no fixed maximum number of rows.

You can drag a REAL database into the Project Window rather than using GetFolderItem.

Example

The following example opens an existing REAL database.

```
Dim dbFile as FolderItem
Dim db as Database
dbFile = GetFolderItem("addresslist.rdb")
db = OpenREALDatabase(dbFile)
If db = Nil then
  Beep
  MsgBox "The database couldn't be opened."
else
  //continue with database operations
end if
```

See Also

NewREALdatabase, Open4DDatabasebyADSP, Open4DDatabasebyTCPIP, OpenCSVCursor, OpenDBFCursor, OpenDTFDatabase, OpenODBCDatabase, OpenOracleDatabase, OpenOpenBaseDatabase, OpenPostgreSQLDatabase, Select4DDatabaseByADSP, Select4DDatabaseByTCPIP, SelectODBCDatabase functions; Database, RecordSet REALDatabase classes.

OpenURLMovie Function

Opens a movie at the specified URL.

Syntax

***result*=OpenURLMovie(URL)**

Part	Type	Description
result	Movie	Movie object
URL	String	URL that specifies the location of the QuickTime movie.

Note

If the user has QuickTime 4 (or above) installed, OpenURLMovie supports streaming video.

Example

The following example loads a streaming video and assigns it to the Movie property of a MoviePlayer control:

```
MoviePlayer1.movie=OpenURLMovie("http://www.apple.com/trailers/miramax/frida.html")
```

Note: Apple Computer, Inc. may remove this movie at their discretion.

See Also

MoviePlayer control; Movie class.

Operator_ Keyword

Enables you to perform comparison and arithmetic operations on custom classes.

Syntax

***result*=Operator_*operation*(ob)**

Part	Type	Description
result	*Type*	The type of the custom class to which the arithmetic operation is being added or a number or string that indicates the result of a comparison operation when Compare is used as *operation*.
ob	*Type*	The type of the custom class to which the comparison or arithmetic operation is being added.
operation	Literal String	Desired comparison or arithmetic operation

Notes

The **Operator_** keyword represents a set of reserved words that enable you to implement the standard arithmetic and comparison operators with custom classes. The supported operators and keywords are shown below:

Operator	Keyword
+	Operator_Add Operator_AddRight
-	Operator_Subtract Operator_SubtractRight
*	Operator_Multiply Operator_MultiplyRight
/	Operator_Divide Operator_DivideRight
\	Operator_IntegerDivide Operator_IntegerDivideRight
Mod	Operator_Modulo Operator_ModuloRight
=, <, >, <=, >=	Operator_Compare
(negation)	Operator_Negate
(convert)	Operator_Convert

For example, to enable you to add two instances of a custom class together using the '+' operator, you would add a function called 'Operator_Add' to the class. The addition operator, +, is not defined for the class until you do this.

Here is an example. Suppose you had a QueryResult object that stored a list of found records and you wanted to combine it with another QueryResult by defining a custom addition operator. Define the Operator_Add keyword using a function such as this:

```
Function Operator_Add( rval As QueryResult ) As QueryResult
  Dim result As QueryResult
  Dim item As FoundRecord
  result = New QueryResult
  For Each item In FoundRecords
    result.FoundRecords.Append item
  Next
  For Each item In rval.FoundRecords
    result.FoundRecords.Append item
  Next
  Return result
End Function
```

Once this definition is in place, this function will be called when you add two QueryResult objects.

All of these operator methods define Self as the left operand and pass the right operand as a parameter. Sometimes this isn't enough; if you had a vector class and you wanted to allow multiplication by a scalar, you could define an Operator_Multiply method like this:

```
Function Operator_Multiply(opB As Double) As Vector
```

This would let you write a expression such as:

```
vectorA = VectorB * 2.5
```

But you could not write this expression:

```
vectorA = 2.5 * VectorB
```

For these situations, there is an additional set of operators. They are identical to the ordinary arithmetic operators, but they reverse the order: Self is now the right operand and the left operand is passed as the parameter.

```
Operator_AddRight(opA As Type) As Type
Operator_SubtractRight(opA As Type) As Type
Operator_MultiplyRight(opA As Type) As Type
Operator_DivideRight(opA As Typ ) As Type
Operator_IntegerDivide(opA As Type) As Type
Operator_ModuloRight(opA As Type) As Type
```

The ordinary methods are always preferred; REALbasic uses the Right versions only if there is no legal left version.

There are two kinds of conversion operators. The Convert To operator has no parameters and returns a value:

```
Operator_Convert() As Type
```

This allows you to use the object anywhere the return type is legal. You can overload this operator with as many different return types as you like, so long as it's always clear which conversion is intended. You can't convert to both an integer and a double, for example, because both would be legal in the same situations.

The other conversion operator is a "convert from." This is like a fusion of the C++ copy constructor and assignment operator. It has one parameter and does not return a value:

```
Operator_Convert(val As Type)
```

In the "convert from" example, an existing object instance was converted into another type. With the From operator, a new instance is created and the appropriate conversion operator is called to initialize it. When an object instance is created by conversion, no constructor is called — the conversion operator takes its place.

If both types of conversion apply in a given situation, REALbasic gives the "from" conversion priority.

If you use an object in an expression and it does not define an appropriate operator, the compiler will give you an undefined operator error.

Or Operator

Used to perform a logical comparison between two Boolean expressions.

Syntax

result=expression1* Or *expression2

Part	Description
result	A Boolean value.
expression1	Any valid Boolean expression.
expression2	Any valid Boolean expression.

Notes

If either expression evaluates to True, then *result* is True. If both expressions evaluate to False then *result* is False.

Example

This example uses the **Or** operator to perform logical comparisons.

```
Dim a As Boolean
Dim b As Boolean
Dim c As Boolean
Dim d As Boolean
a=True
b=False
d=a Or b //Returns True
d=b Or c //Returns False
```

See Also

And, Not operators.

OutOfBoundsException Error

Occurs when code tries to access an array element that doesn't exist.

Super Class

RuntimeException class

Notes

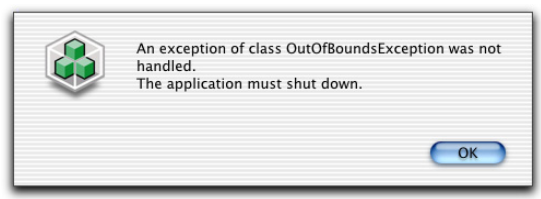
An **OutOfBoundsException** error occurs when your code uses an array index that is out of range. This occurs, for example, if you forget that arrays in REALbasic are zero-based and you make an error when you try to access the last element.

Examples

This example is supposed to display the fonts installed on the user's computer in a ListBox. It generates an OutOfBoundsException because the loop should be from 0 to nFonts-1 rather than from 1 to nFonts. When the loop gets to nFonts, the value of i is out of range and the error occurs.

```
Dim i, nFonts as Integer
nFonts=FontCount
For i=1 to nFonts //don't do this!
 ListBox1.addrow(Font(i))
Next
```

If you run this code in a built application, you will see a generic error message when i reaches the value of nFonts:

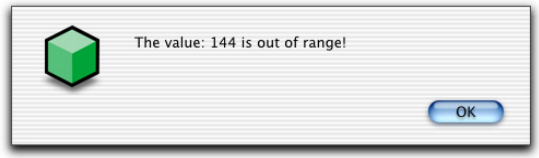


The application quits when the user accepts this dialog.

You can handle the exception by adding the following code:

```
Dim i, nFonts as Integer
nFonts=FontCount
For i=1 to nFonts //don't do this!
 ListBox1.addrow(Font(i))
Next
Exception err
If err IsA OutOfBoundsException then
 MsgBox "The value: "+Str(i)+" is out of range!"
end if
```

When this code runs in a built application, the user sees the following dialog box.



The application does not quit when the user accepts the dialog box.

See Also RuntimeException class, NilObjectException error, IllegalCastException error, StackOverFlowException error, TypeMismatchException error, Function statement, Raise statement, Nil function, Exception block.

OutOfMemoryException Error

Raised in certain cases when you run out of memory.

SuperClass RuntimeException class.

Notes The **OutOfMemoryException** Error may be raised when an operation cannot be completed due to insufficient memory.

See Also RuntimeException class, NilObjectException error, IllegalCastException error, StackOverFlowException error, TypeMismatchException error, Function statement, Raise statement, Nil function, Exception block.

Oval Control 

Draws an oval.

Super Class RectControl

Because this is a RectControl, see the RectControl for other properties and events that are common to all RectControl objects.

Properties

Name	Type	Description
BorderColor	Color	The color of the oval's border.
BorderWidth	Integer	The width of the oval's border in pixels.
FillColor	Color	The interior color of the oval.

Events

Name	Parameters	Description
MouseDown	x as Integer, y as Integer	The mouse button was pressed inside the Oval region at the location passed in to x,y. Return True if you are going to handle the MouseDown.
MouseDrag	x as Integer, y as Integer	The mouse button was pressed inside the Oval and moved (dragged) at the location local to the control passed in to x,y. This event will not occur unless you return True in the MouseDown event.
MouseUp	x as Integer, y as Integer	The mouse button was released inside the Oval region at the location passed in to x,y. This event will not occur unless you return True in the MouseDown event.

See Also

RectControl class.

OvalShape Class

Used for drawing (two-dimensional) ellipses in a vector graphics environment.

Super Class RectShape

Properties

Name	Type	Description
Segments	Integer	The number of polygon sides to use when approximation is needed. Zero means it's up to the renderer.

Example

Place the following method in the MouseDown event handler of a window. It will draw an oval in the window when the user presses the mouse button.

```
Dim o as OvalShape
o=New OvalShape
o.width=60
o.height=30
o.Fillcolor=RGB(127,127,255)

Graphics.DrawObject o,x,y
```

See Also

ArcShape, CurveShape, FigureShape, FolderItem, Group2D, Graphics, Picture, PixmapShape, RectShape, RoundRectShape, StringShape classes.

PagePanel Control

Similar to the TabPanel control, but does not include tabs (or any other visual indicator) for navigating to different pages. The TabPanel is subclassed from the **PagePanel**.

SuperClass

RectControl

Properties

Name	Type	Description
Value	Integer	The number of the currently selected page. The first page is number zero.

Events

Name	Parameters	Description
Change		The frontmost page has changed.

Notes

The **PagePanel** control allows you to add controls to pages, just as you add controls to tab panels in a TabPanel. The difference is that the **PagePanel** control itself is not

visible in your built application—only the controls on the currently selected page are visible. You change the displayed page by changing the value of the Value property. You can embedded PagePanels within PagePanels.

In the IDE, you can add, insert, delete, reorder and navigate from page to page, just like a TabPanel. When building an interface using the **PagePanel**, you can take advantage of the Control Hierarchy, discussed in the section on the Control object.

See Also TabPanel control; Control object.

Parameter and Default Value Must be of the Same Type Error

You used a value of an incorrect data type as a default value. When you create a method in the New Method dialog box, you can optionally set a default value for each of the parameters. Of course, the data type of the default value must match the data type of the parameter.

Example Consider the following parameter declaration:

```
s as String = 15
```

Since s is declared as a String, its default value must be a String. 15 isn't.

Parameters are not Compatible with this Function Error

You passed parameters of an incorrect data type to a function.

Example The RGB function expects integer values:

```
Dim c as Color
c=RGB ("red","green","blue")
```

ParseDate Function

Parses a date passed as a string and creates a Date object.

Syntax

***result*=ParseDate(*Text*, ByRef *ParsedDate*)**

Part	Type	Description
result	Boolean	Returns True if *Text* was successfully parsed.
Text	String	The date string to be parsed.
ParsedDate	Date	After successful call, contains the parsed date as a Date object.

Notes

The *Text* parameter must be a valid string representation of a date. You can use the slash, the dash, the period, or the space to separate the Month, Day, and Year. The following are valid string representations: 12/14/1341, 12-14-1341, 12.14.1341, 14Dec1341, 14.Dec.1341, and 14 Dec. 1341.

If you use only the last two digits of the year, the current century is assumed.

ParseDate does not parse a the Time value of a date, if present.

Example

The following code displays the parsed date in a message box in the Abbreviated Date format. You can use any Date property to return corresponding values, e.g., the Day, Month, and Year properties hold the day, month, and year of the passed date.

```
Dim theDate as Date
Dim theTrueBool as Boolean
theDate=New Date
thetruebool=ParseDate("14Dec1341",theDate)
If theTrueBool then
 MsgBox thedate.AbbreviatedDate
else
 MsgBox "Invalid Date format!"
end if
```

See Also

Date class, Database class examples.

Picture Class

A **Picture** object contains a picture.

Super Class

Object

Properties

Name	Type	Description
Depth	Integer	The depth (bytes) of the picture. A depth of zero indicates that the picture has no bitmap at all, but is for vector graphics.
ImageCount	Integer	Number of indexed images in the picture (QuickTime 4 and above only).
IndexedImage	**Picture**	Returns i^{th} image in the picture (QuickTime 4 and above only). Parameter is Index as Integer.
Graphics	Graphics	The actual picture.
Height	Integer	The height (in pixels) of the picture.
HorizontalResolution	Integer	Horizontal resolution of the picture. Established when you use the OpenAsPicture method of the FolderItem class and written out when using the SaveAsPicture method (of the FolderItem class). When creating a picture, the default resolution is 72 (pixels).
Mask	**Picture**	Allows you to access a 256-gray mask that controls the transparency of the picture.
Objects	Group2D	A set of vector graphics associated with the picture (optional).
RGBSurface	RGBSurface	Provides pixel-level access to the picture's bitmap. Used only for pictures built with NewPicture with a depth of 16 or 32, otherwise returns Nil. Returns RGBSurface object.
Transparent	Integer	Controls transparency of the white 'layer' of the picture. 0 - Not transparent. 1 - White is transparent.
VerticalResolution	Integer	Vertical resolution of the picture. Established when you use the OpenAsPicture method of the FolderItem class and written out when you use the SaveAsPicture method (of the FolderItem class).When creating a picture, the default resolution is 72 (pixels).
Width	Integer	The width (in pixels) of the picture.

Notes

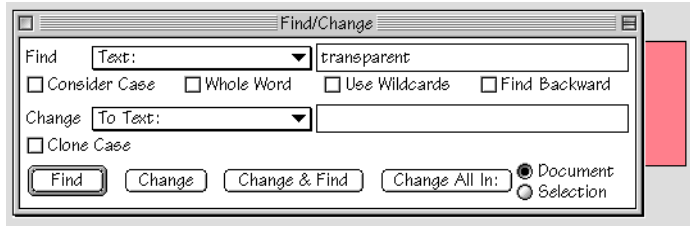
A **Picture** object is created by adding a picture to the Project Window, by calling the OpenAsPicture or OpenAsVectorPicture methods of a FolderItem object, by calling the NewPicture function, or by using the New operator. When you use the New operator,

you must pass the same parameters as when using NewPicture: Width as Integer, Height as Integer, and Depth as Integer. Width and Height are in pixels and the values of Depth may be 0, 1, 2, 4, 8, 16, or 32. If you use the New operator but run out of memory, an OutOfMemoryException error will be raised. If you specify a depth of zero REALbasic creates a picture with no pixel map at all, but with a preinitialized Objects property. Use this option for creating vector graphics pictures via the set of Object2D subclasses.

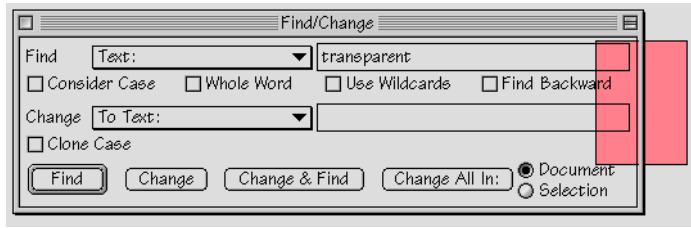
Picture objects have no methods or events.

The Transparent property works only if making the color white transparent produces the desired effect. In the example shown below, the picture is the screen shot of the Find dialog box. Making the white 'layer' transparent makes the picture object transparent.

Transparent=0



Transparent=1



Otherwise, you need to work with the Mask property. For example, if you have a picture, myPicture, and a custom mask, myMask, stored in the Project Window, the code:

```
myPicture.Mask.Graphics.DrawPicture myMask,0,0
```

will assign the custom mask to the Mask property of myPicture.

See Also

NewPicture function, RGBSurface object, Graphics class.

PixmapShape Class

Used to place a picture in a vector graphics environment.

Super Class

Object2D

Properties

Name	Type	Description
Image	Picture	The source image to use.
SourceHeight	Integer	Height (pixels) of source image, from *SourceTop*, *SourceLeft*. Used to define a subregion of image to use.
SourceLeft	Integer	Number of pixels from left edge of source image. Used to define subregion of image to draw.
SourceTop	Integer	Number of pixels form top edge of source image. Used to define subregion of image to draw.
SourceWidth	Integer	Width (pixels) from SourceLeft of image to draw. Used to define a subregion of image to use.

Constructor

Name	Parameters	Description
PixmapShape	Image as Picture	Initializes object to entire source image.

Example

The following method in the Action event of a PushButton draws a red oval in the parent window.

```
Dim p as Picture
Dim px as PixmapShape

p=New Picture(240,200,32)
p.graphics.foreColor = &CFF0000
p.graphics.fillOval 0,0,240,200

px=New PixmapShape(p)
px.rotation = 45/57.2958  //45 Degrees in radians

Graphics.drawObject px,150,150
```

See Also

ArcShape, CurveShape, FigureShape, FolderItem, Group2D, Graphics, Object2D OvalShape, Picture, RectShape, RoundRectShape, StringShape classes.

Placard Control 

Used to add a placard to a window. A placard is a control that you can use as an information display or as background fill for a control area. **Placards** have three states: normal, pressed, and disabled.

Perhaps the most familiar use of the **Placard** control is as a small information panel, often placed at the bottom of a window to the left of the horizontal scroll bar.

Super Class

RectControl

Because this is a RectControl, see the RectControl for other properties and events that are common to all RectControl objects.

Properties

Name	Type	Description
Enabled	Boolean	If True, draws a shaded border around the object; if False, the border is a line drawing and the placard is disabled.
Value	Boolean	If *Enabled* is True, *Value* shades the interior of the placard (Pressed state). If *Enabled* is True and *Value* is False, the interior is unshaded (Normal state).

Events

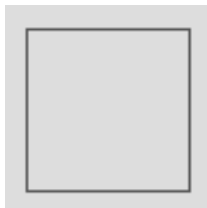
Name	Parameters	Description
MouseDown	X as Integer, Y as Integer	The mouse button has been pressed inside the window at the x,y local coordinates passed.
MouseUp	X as Integer, Y as Integer	The mouse button has been released inside the window at the x,y local coordinates passed.
MouseDrag	X as Integer, Y as Integer	The mouse button was pressed inside the window and moved (dragged) at the location local to the window passed in to x,y. This event will not occur unless you return True in the MouseDown event.

Note

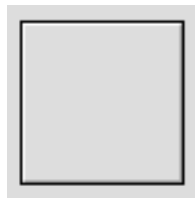
On Mac OS X, you can solve some display problems with Placards by turning on the parent window's Composite property. For an example, see the Notes on the Window class.

Examples

The following illustrations show a **Placard** control in its various states:



Disabled



Value=False

Value=True

See Also

RectControl class.

Pop Method

Removes the last element from an array and returns its value.

Syntax

***result*=*array*.Pop**

Part	Description
result	The element of *array* that was removed.
array	Required. The array from which the last value will be popped.

Notes

The Append and **Pop** methods can be used together to treat an array as a stack.The Append method pushes a new element onto the array/stack and the Pop method removes the last element from the array/stack and returns its value.

Example

This is a directory-crawling routine that performs a depth-first traversal without using recursion:

```
Dim ItemsToInspect(0) As FolderItem
Dim item As FolderItem, ctr As Integer
ItemsToInspect(0) = directory
While Ubound(itemsToInspect) >= 0
 item = itemsToInspect.Pop
 If item.Directory Then
  For ctr = 1 To item.Count
   itemsToInspect.Append item
  Next
 End If
Wend
```

See Also

Append, Insert, Remove, Ubound methods.

POP3SecureSocket Class

Used to retrieve messages on a POP3 mail server using SSL or TLS encryption.

Super Class

SSLSocket

Properties

Name	Type	Description
Username	String	The username to use for authentication when connecting to the mail server.
Password	String	The password to use for security when connecting to the mail server.
EncryptPassword	Boolean	If True, the password is encrypted when being sent to the mail server.

Methods

Name	Parameters	Description
CheckServerConnection		Sends a "NOOP" command to the mail server. This is simply a command that asks the server to reply. This can be useful to check that the mail server is still responding and also tells the mail server that you are still connected if there has been no activity for a long period of time.
Connect		Connects to the mail server and logs in with the Username and Password.
CountMessages		Asks the server for the number of messages in the mailbox.
DeleteMessage	Index as Integer	Tells the mail server to delete the specified message.
DisconnectFromServer		Disconnects from the mail server. This sends a "QUIT" command to the mail server and waits for it to close the connection.
ListMessages	[Index as Integer]	Requests a message listing. This list consists of the message index and the size of the message. If no index is passed, it gets the entire list from the server. If a specific index is passed it will return just the *index* message and size of the message.
RetrieveLines	Index as Integer, LineCount as Integer	Returns the specified number of lines of a message. The mail server will return the first *LineCount* of lines that exist in the message you are requesting via the *Index* parameter. If *LineCount* is zero, then the mail server returns only the headers for the message.

Name	Parameters	Description
RetrieveMessage	Index as Integer	Reads the entire message specified by *Index*.
RollbackServer		Resets the mail server to the state that it was when you logged in. Can be used to Undo deletions that occur by accident. The changes aren't committed until the connection is closed. RollbackServer will roll back changes that have not yet been committed.
SendServerCommand	Command as String	Sends the command specified by *Command* to the mail server. This is useful when you need to send a command that REALbasic doesn't yet support.

Events

Name	Parameters	Description
ConnectionEstablished		Occurs when a connection has been established.
Disconnected		Occurs when the connection has been terminated.
ListReceived	List as String	Executes when the ListMessages is called. List contains the message listing.
LoginSuccessful		Executes when the login process initiated by calling the Connect method is complete.
MessageCount	Count as Integer	Executes when the mail server replies to a CountMessages call and contains the number of messages in the mailbox.
MessageDeleted	Index as Integer	Executes when the mail server replies to a DeleteMessage call and contains the index number of the deleted message.
MessageReceived	Index as Integer, Message as emailMessage	Executes when a message has been received from the mail server, in response to a call to RetrieveMessage. Index contains the index number of the retrieved message and the message contents is in Message.
RollbackSuccessful		Executes in response to a call to RollbackServer and indicates that the state of the mailbox has been reset.
ServerAvailable		Executes when the mail server has replied to a call to CheckServerConnection and indicates that the mail server has replied to the call.

Name	Parameters	Description
ServerError	ErrorCode as Integer, ErrorMessage as String, MessageID as Integer	Executes when a protocol-related error occurs. The error codes are as follows: 0 - Unknown Error Message 1 - Incorrect Password 2 - IncorrectUsername 3 - Delete Message Failed 4 - List Messages Failed 5 - Retrieve Lines Failed 6 - Retrieve Message Failed
TopLinesReceived	Index as Integer, data as EmailMessage	Executes in response to a call to RetrieveLines. Index contains the index number of the partial message being retrieved and data contains the requested lines of the message.

Notes

The **POP3SecureSocket** class is identical to the POP3Socket class, except that it is derived from the SSLSocket class instead of the TCPSocket class. This enables you to send secure email by setting the Secure property of the SSLSocket class.

Example

The following code in the MessageReceived event handler places the body of an email message in an EditField.

```
Sub MessageReceived(ID as Integer, email as emailMessage)
 Dim s as String
 s=email.bodyHTML
 If s="" then
  s=email.bodyPlainText
 end
EditField1.text=s
End Sub
```

See Also

SMTPSocket, SMTPSecureSocket, POP3Socket, EmailMessage, HTTPSocket, TCPSocket, SSLSocket, SocketCore classes.

POP3Socket Class

Used to retrieve and manage messages on a POP3 mail server.

Super Class

TCPSocket

Properties

Name	Type	Description
Username	String	The username to use for authentication when connecting to the mail server.
Password	String	The password to use for security when connecting to the mail server.
EncryptPassword	Boolean	If True, the password is encrypted when being sent to the mail server.

Methods

Name	Parameters	Description
CheckServerConnection		Sends a "NOOP" command to the mail server. This is simply a command that asks the server to reply. This can be useful to check that the mail server is still responding and also tells the mail server that you are still connected if there has been no activity for a long period of time.
Connect		Connects to the mail server and logs in with the Username and Password.
CountMessages		Asks the server for the number of messages in the mailbox.
DeleteMessage	Index as Integer	Tells the mail server to delete the specified message.
DisconnectFromServer		Disconnects from the mail server. This sends a "QUIT" command to the mail server and waits for it to close the connection.
ListMessages	[Index as Integer]	Requests a message listing. This list consists of the message index and the size of the message. If no index is passed, it gets the entire list from the server. If a specific index is passed, it will return just the *index* message and size of the message.
RetrieveLines	Index as Integer, LineCount as Integer	Returns the specified number of lines of a message. The mail server will return the first *LineCount* of lines that exist in the message you are requesting via the *Index* parameter. If *LineCount* is zero, then the mail server returns only the headers for the message.

Name	Parameters	Description
RetrieveMessage	Index as Integer	Reads the entire message specified by *Index*.
RollbackServer		Resets the mail server to the state that it was when you logged in. Can be used to Undo deletions that occur by accident. The changes aren't committed until the connection is closed. RollbackServer will roll back changes that have not yet been committed.
SendServerCommand	Command as String	Sends the command specified by *Command* to the mail server. This is useful when you need to send a command that REALbasic doesn't yet support.

Events

Name	Parameters	Description
ConnectionEstablished		Occurs when a connection has been established.
Disconnected		Occurs when the connection with the server has been lost.
ListReceived	List as String	Executes when the ListMessages is called. *List* contains the message listing.
LoginSuccessful		Executes when the login process initiated by calling the Connect method is complete.
MessageCount	Count as Integer	Executes when the mail server replies to a CountMessages call and contains the number of messages in the mailbox.
MessageDeleted	Index as Integer	Executes when the mail server replies to a DeleteMessage call and contains the index number of the deleted message.
MessageReceived	Index as Integer, Message as emailMessage	Executes when a message has been received from the mail server, in response to a call to RetrieveMessage. Index contains the index number of the retrieved message and the message contents is in Message.
RollbackSuccessful		Executes in response to a call to RollbackServer and indicates that the state of the mailbox has been reset.
ServerAvailable		Executes when the mail server has replied to a call to CheckServerConnection and indicates that the mail server has replied to the call.
ServerCommandReply	Command as String, Data as String	Executes in response to a call to SendServerCommand and contains the mail server's response to the command passed.

Name	Parameters	Description
ServerError	ErrorCode as Integer, ErrorMessage as String, MessageID as Integer	Executes when a protocol-related error occurs. The error codes are as follows: 0 - Unknown Error Message 1 - Incorrect Password 2 - IncorrectUsername 3 - Delete Message Failed 4 - List Messages Failed 5 - Retrieve Lines Failed 6 - Retrieve Message Failed
TopLinesReceived	Index as Integer, data as EmailMessage	Executes in response to a call to RetrieveLines. Index contains the index number of the partial message being retrieved and data contains the requested lines of the message.

See Also

SMTPSocket, EmailMessage, HTTPSocket, TCPSocket, SocketCore classes.

PopupArrow Control

Used to add a popup arrow to a window.

Super Class

RectControl

Because this is a RectControl, see the RectControl for other properties and events that are common to all RectControl objects.

Properties

Name	Type	Description
Facing	Integer	Controls direction and size of the PopupArrow 0—East 1—West 2—North 3—South 4—Small East 5—Small West 6—Small North 7—Small South

Example

The following line of code changes the size and direction of a **PopupArrow**:

```
popuparrow1.facing=3
```

See Also

RectControl class.

PopupMenu Control

The popup menu control.

Super Class

RectControl

Because this is a RectControl, see the RectControl for other properties and events that are common to all RectControl objects.

Properties

Name	Type	Description
Bold	Boolean	Applies the bold style to all items in the list.
DataField	String	Relevant only if the PopupMenu is used in conjunction with a DataControl to display the contents of fields in a database table. Name of a Field in the table referenced by DataControl.
DataSource	DataControl	The DataControl that references a table in a database whose fields are displayed. Use the DataField property to link a field to be displayed in a PopupMenu. In the IDE, a popup menu of field names will appear. When setting in code, it takes a String. Assign it to the Name property of a DataControl. Note: Database fields can also be displayed using EditFields, ListBoxes, and StaticText controls.
InitialValue	String	A list of the default items separated by carriage returns. This property is unreadable at runtime because the list is removed from memory once the control is created. You can set a default value for the control by populating the pop-up with *InitialValue* and setting the default value with *ListIndex*.
Italic	Boolean	Applies the italic style to all items in the list.
List	Array as String	The list of items.
ListCount	Integer	The number of items in the list.
ListIndex	Integer	Used to get or set the selected item number. The first item is numbered zero. A value of -1 means no selection. May be set using the Properties window to set default value of PopupMenu control. If you use the InitalValue property to populate the PopupMenu control and set a value of ListIndex that corresponds to a default item, that item will be displayed in the Design environment.
Text	String	The text of the currently selected item.
TextFont	String	Name of the font used to display the button text.

Name	Type	Description
TextSize	Integer	Size of the font used to display the button text.
Underline	Boolean	Applies the underline style to all items in the list.

Events

Name		Description
Change		The selected item has changed.
GotFocus		(Win32 only) The control has received the focus. If the user tabbed to select it, the current item is highlighted.
LostFocus		(Win32 only) The control has lost the focus.
MouseDown	x as Integer, y as Integer	The mouse button was pressed inside the control's region at the location passed in to x,y. Return True if you are going to handle the MouseDown.
MouseUp	x as Integer, y as Integer	The mouse button was released inside the control's region at the location passed in to x,y. This event will not occur unless you return True in the MouseDown event.

Methods

Name	Parameters	Description
AddRow	Item as String	Appends Item in a new row to the end of the list.
AddSeparator		Adds a separator line to the popup menu.
DeleteAllRows		Deletes all rows in the list.
InsertRow	RowNumber as Integer, Item as String	Creates a new row at RowNumber (moving the existing row down).
RemoveRow	RowNumber as Integer	Deletes the specified row.
RowTag	Row as Integer	A 'hidden' identifier associated with the item identified by *Row*. See the example in Chapter 10 of the *User's Guide* of a database created using object bindings for an example of how RowTag is used to store the ID associated with each PopupMenu item. Returns a Variant. If you want to compare the value of RowTag to another value, you must first convert the value of RowTag to that data type.

Examples

This code in the Open event handler populates a popup menu and sets the initial value to the current month:

```
Dim s as String
Dim i,last as Integer
Dim d as Date
d=New Date
s="January,February,March,April,May,June, July,
    August,September,October,November, December"
last=CountFields(s,",")
For i=1 to last
 me.addRow NthField(s,",",i)
Next
me.ListIndex=d.Month-1
```

Examine the value of ListIndex in the Change event handler to determine which item was selected.

This example adds an item to PopupMenu1.

```
PopupMenu1.addrow "October"
```

This example populates the RowTag identifier with a sequence number:

```
Dim i,nItems as Integer
nItems=Me.ListCount-1
For i=0 to nitems
 Me.rowTag(i)=i
Next
```

Since RowTag is a Variant, you must first convert it to an Integer if you want to compare it to another integer. Do this with a simple assignment statement such as:

```
Dim RecID as Integer
RecID=PopupMenu1.Rowtag(1)
```

Then compare RecID to another Integer.

This example opens a new window when an item is chosen.

```
Sub Change()
Dim w As ListEditorWindow
 If PopupMenu1.text="Edit List..." Then
   w=New ListEditorWindow
 End If
End Sub
```

Changing the selected item in a PopupMenu:

```
PopupMenu1.listIndex=3
```

Displaying the RowTag of the selected menu item:

```
EditField1.text=popupmenu1.rowtag(popupMenu1.listindex)
```

See Also BevelButton, ContextualMenu controls; RectControl class.

PostgreSQLDatabase Class

Used to open a PostgreSQL database. Please use this class instead of OpenPostgreSQLDatabase.

Super Class Database

Properties

Name	Type	Description
Port	Integer	The port of the PostgreSQL database to connect to.

Notes In order to use this class, you must install the appropriate database plug-in in your plugins folder.

The set of database plug-ins is included on the REALbasic CD, but you may find more recent versions at the REAL Software web site, www.realsoftware.com.

See Also Database class.

Pow Function

Returns the value specified raised to the power specified.

Syntax ***result*=Pow (*value*, *power*)**

Part	Type	Description
result	Double	*Value* raised to *power*.
value	Double	The value you want to raised to *power*.
power	Double	The power *value* is raised to.

Examples

This example uses the **Pow** function to return four raised to the power of seven.

```
Dim d As Double
d=Pow(4,7) //returns 16384
```

PreferencesFolder Function

Used to access the Preferences folder in the System folder. On Windows, it accesses the c:\Documents and Settings*username*\Local Settings\Application Data\ directory if available. On Windows 95/98 it accesses the Windows directory.

Syntax

***result*=PreferencesFolder**

Part	Type	Description
result	FolderItem	A FolderItem that represents the Preferences Folder or Application Data directory.

Notes

Use the **PreferencesFolder** function to access the Preferences folder in the user's System Folder.

If your application has user-defined preferences, it is a good idea to store the preference values in a file and place it in the Preferences folder. This allows the user to reinstall your application if necessary and retain their current preferences.

The **PreferencesFolder** function provides a way to access the Preferences folder that will work under different language systems, and will continue to work even if Apple reorganizes the System folder.

Examples

See the AppleMenuFolder function for an example.

See Also

AppleMenuFolder, ApplicationSupportFolder, ControlPanelsFolder, DesktopFolder, ExtensionsFolder, FontsFolder, StartupItemsFolder, SystemFolder, TemporaryFolder, TrashFolder functions.

PrefsMenuItem Class

A menu item derived from the **PrefsMenuItem** class is automatically moved to the application menu under Mac OS X (or the REALbasic menu when running within the IDE). Under other operating systems, such a menu item stays where you put it in the Menu editor.

Super Class

Object

Properties

Name	Type	Description
BalloonHelp	Message as String	Message that appears when Balloon Help is on and the user holds the mouse over the menu item while the item is enabled.
Bold	Boolean	Indicates if the menu item should appear in bold.
Checked	Boolean	Indicates whether or not the menu item is checked.
CommandKey	String	The Macintosh keyboard shortcut for the menu item. Keyboard shortcuts for Windows are set using Constants. See the section "Using Constants to Add Keyboard Shortcuts to Menus and Menu Items" in the *User's Guide*.
DisabledBalloonHelp	Message as String	Message that appears when Balloon Help is on and the user holds the mouse over the menu item while the item is disabled.
Enabled	Boolean	Indicates whether or not the menu item is enabled. Menu items based on this class are enabled by default. You can disable a menu item either by assigning the Enabled property a value of False.
Italic	Boolean	Indicates if the menu item should appear in italic.
Name	String	The name of the menu item.
Submenu	Boolean	Indicates that the menu item is a submenu.
Tag	String	A 'hidden' text string associated with the menu item. The tag is accessible via code when the user chooses the menu item but, unlike the Text property, is not displayed in the menu.
Text	String	The text of the menu item.
Underline	Boolean	Indicates if the menu item should appear underlined.
Visible	Boolean	Indicates whether or not the menu item is visible.

Methods

Name	Parameters	Description
Close		Removes dynamically created menu items.

Name	Parameters	Description
Enable		Sets the Enabled property of the menu item to True. Call Enable only within an EnableMenuItems event handler.

Notes

This class is obviously for creating the Preferences menu item for your application. Create the menu item under the menu that you want to use for the Windows and Mac OS "classic" builds and then use the Properties window to change its super class to **PrefsMenuItem**. Otherwise, set up the menu item normally. In your Mac OS X build, it will magically move to the built application's menu.

According to Apple user interface guidelines, ⌘-, (comma) is the recommended keyboard shortcut for an application's Preferences menu item.

See Also

MenuItem class.

PrinterSetup Class

Used to get and set the page setup settings.

Super Class

Object

Properties

Name	Type	Description
Height	Integer	The height of the printable area on the page in pixels. The height is the PageHeight minus the margins.
HorizontalResolution	Integer	The horizontal resolution in dots per inch of the output device (usually a printer).
Left	Integer	The left origin of the printable area. This should always be zero.
MaxHorizontalResolution	Integer	Set to the maximum horizontal resolution at which you wish to print. The default value is 72. You can change it to the maximum printer resolution you're prepared to handle or -1 for the highest possible resolution of the output device. After calling PageSetupDialog, OpenPrinter, or OpenPrinterDialog, the HorizontalResolution property will automatically be set to the highest resolution supported by the printer driver, within your specified constraints. Printing will then occur at that resolution. Will not affect existing code.

Name	Type	Description
MaxVerticalResolution	Integer	Set to the maximum vertical resolution at which you wish to print. The default value is 72. You can change it to the maximum printer resolution you're prepared to handle or -1 for the highest possible resolution of the output device. After calling PageSetupDialog, OpenPrinter, or OpenPrinterDialog, the VerticalResolution property will automatically be set to the highest resolution supported by the printer driver, within your specified constraints. Printing will then occur at that resolution. Will not affect existing code.
PageHeight	Integer	The height of the page in pixels.
PageLeft	Integer	The offset distance (in pixels) from the left margin of the printable area to the left edge of the physical page, i.e., the left margin.
PageTop	Integer	The offset distance (in pixels) from the top margin of the printable area to the top edge of the physical page, i.e., the top margin.
PageWidth	Integer	The width of the page in pixels.
SetupString	String	A value that represents all of the other properties. When the user clicks OK in the Page Setup dialog box, this value will be populated. You can then store this value to store the users Page Setup settings. Assigning one of these stored values to this property will then update all of the other properties restoring the page setup settings.
Top	Integer	The top origin of the printable area. This should always be zero.
VerticalResolution	Integer	The vertical resolution in dots per inch of the output device (usually a printer).
Width	Integer	The width of the printable area on the page in pixels. The width is the PageWidth minus the margins.

Methods

Name	Parameters	Description
PageSetupDialog	[window] as Window	Displays the standard Page Setup dialog box. If the SetupString property has been populated before this method is called, the Page Setup dialog box will reflect the settings stored in the SetupString property. After the user clicks the OK button to close the Page Setup dialog box, all of the PrinterSetup properties will be updated to reflect the settings the user chose. Takes an optional parameter, *window*. If passed, *window* is a sheet window that you wish to use as the Page Setup dialog box instead of the built in dialog. Returns a Boolean. This function returns True if the User clicks OK and False if the user clicks Cancel.

Notes

Passing a PrinterSetup object to the OpenPrinter or OpenPrinterDialog functions will cause the printer to utilize those PrinterSetup object's properties when printing. For example, if the user chose 200% for the scale in the Page Setup dialog box, the printer would automatically print at 200%.

MaxHorizontalResolution and MaxVerticalResolution

These properties enable you to print at higher resolutions than 72 dpi. In general, you will need to scale the material being printed (and perhaps the size of the controls) to get the desired results. For example, if you are printing styled text in an EditField using DrawBlock, you will need to make commensurate changes to the font size(s) to get WYSIWYG output. Doubling the resolution will require that you double the font size; otherwise you will get text that is half the size of the screen font.

Examples

This example displays the Page Setup dialog box then stores the settings the user chose in a variable:

```
Dim settings as String
Dim PageSetup as PrinterSetup
PageSetup=New PrinterSetup
If PageSetup.PageSetupDialog Then
  settings=PageSetup.SetupString
End If
```

This example restores the page setup settings stored in a variable called "settings" and then displays the Page Setup dialog box with those settings:

```
Dim PageSetup as PrinterSetup
PageSetup=New PrinterSetup
PageSetup.SetupString=settings
If PageSetup.PageSetupDialog Then
  settings=PageSetup.SetupString
End If
```

This example displays the Page Setup dialog box and then passes the settings the user chose to the OpenPrinterDialog function. It then prints a sample string:

```
Dim g as Graphics
Dim p as PrinterSetup
p=New PrinterSetup
If p.PageSetupDialog then
  g=OpenPrinterDialog(p)
  if g<> Nil then
    g.DrawString "Hello World", 50,50
  End if
End if
```

This example displays the Page Setup box and then displays the page size, printable area, and margins in StaticText controls. Results, of course, depend on the page size that the user selects. Since PageLeft and PageTop are the horizontal and vertical margins as measured from the printable area rather than the edge of the page, they are negative.

```
Dim settings as String
Dim p as PrinterSetup
p=New PrinterSetup
If p.PageSetupDialog Then
  settings=p.SetupString
End If
staticText1.text="PageLeft="+str(p.pageLeft)
staticText2.text="PageTop="+Str(p.pagetop)
staticText3.text="PageHeight="+Str(p.pageheight)
staticText4.text="PageWidth="+Str(p.pagewidth)
staticText5.text="Height="+Str(p.Height)
staticText6.text="Width="+Str(p.width)
staticText7.text="Computed height="+Str(p.height-2*p.pagetop)
staticText8.text="Computed width="+Str(p.width-2*p.pageleft)
```

See Also

Graphics class, OpenPrinter function, OpenPrinterDialog function, StyledTextPrinter class.

ProgressBar Control

Displays a progress thermometer.

Super Class

RectControl

Because this is a RectControl, see the RectControl for other properties and events that are common to all RectControl objects.

Properties

Name	Type	Description
Maximum	Integer	The maximum value of the progress bar. The maximum value of *Maximum* is 65536.
Value	Integer	The current value of the progress bar.

Events

Name	Parameters	Description
MouseDown	x as Integer y as Integer	The mouse button was pressed inside the ProgressBar at the location passed in to x,y. Returns a Boolean. Return True if you are going to handle the MouseDown.
MouseDrag	x as Integer y as Integer	The mouse button was pressed inside the ProgressBar and moved (dragged) at the location local to the control passed in to x,y. This event will not occur unless you return True in the MouseDown event.
MouseUp	x as Integer y as Integer	The mouse button was released inside the ProgressBar at the location passed in to x,y. This event will not occur unless you return True in the MouseDown event.

Notes

When you want to show progress of an indeterminate length, set the Maximum property to zero to display a "barber pole" rather than a progress thermometer.

Progress Bar

Barber Pole

Example

This line of code sets the maximum value of the progress bar:

```
ProgressBar1.maximum=350
```

See Also

RectControl class.

PushButton Control 

The standard push button.

Super Class

RectControl

Because this is a RectControl, see the RectControl for other properties and events that are common to all RectControl objects.

Properties

Name	Type	Description
Bold	Boolean	Applies the bold style to the button caption.
Cancel	Boolean	If True, the Escape key and Command-Period key sequence are mapped to the button.
Caption	String	The button's text.
Default	Boolean	If True, the default button indicator is added to the button and the Return and Enter keys are mapped to the button.
Italic	Boolean	Applies the italic style to the button caption.
TextFont	String	Name of the font used to display the button caption.
TextSize	Integer	Size of the font used to display the button caption.
Underline	Boolean	Applies the underline style to the button caption.

Methods

Name	Parameters	Description
Push		Causes the button to be visually pushed (clicked), as well as triggering the Action event. Call Push when the user has pressed the keyboard equivalent for the button to give the appropriate visual feedback.
SetFocus		(Win32 only) Sets the focus to the PushButton.

Events

Name	Parameters	Description
Action		The button has been clicked.
GotFocus		(Win32 only) The button has received the focus and has a selection marquee around the text of the Pushbutton.
LostFocus		(Win32 only) The button has lost the focus.
MouseDown	x as Integer, y as Integer	The mouse button was pressed inside the control's region at the location passed in to x,y. Return True if you are going to handle the MouseDown. The Action event will not execute and the state of the object will not change.
MouseUp	x as Integer, y as Integer	The mouse button was released inside the control's region at the location passed in to x,y. This event will not occur unless you return True in the MouseDown event. The return value is ignored.

Note

If the Caption property contains an ampersand character, it will display only if it is preceded by another ampersand character. This is done to make Macintosh and Windows applications behave consistently.

The PushButton's Caption property can show a caption that uses an encoding that doesn't match the application's region code (or while running within the IDE, the IDE's region code). For example, an English application can set a Japanese caption, as long as it first sets the button's TextFont to Osaka.

See Also

BevelButton control; RectControl class.

QuickTime Object

Provides access to certain properties of the QuickTime software installed on the user's system. Accessible only via the System Object.

Properties

Name	Type	Description
Installed	Boolean	Returns True if QuickTime is installed on the user's computer.

Example

This example determines whether QuickTime is installed.

```
If System.QuickTime.Installed then
 Beep
 MsgBox "QuickTime is installed!"
End if
```

See Also

System object.

QT3DAudio Class

A storage container for 3D Audio settings. The properties of this class enable you to store such settings.

Super Class

Object

Properties

Name	Type	Description
ConeAngleCos	Double	Cosine (angle/2) of the attenuation cone, in degrees. The range is from zero to 360.
ConeAttenuation	Double	Amount of attenuation (in decibels) outside the cone.
CPULoad	Boolean	CPU load versus quality: zero is best quality.
Humidity	Double	The humidity (per cent) when the medium is air. The range is from zero to 100.
LocationAzimuth	Double	The range is from -1.57 to 1.57.
LocationDistance	Double	The units are meters and range is from 1.0 to 10.0.
LocationElevation	Double	The units are meters and range is from -3.14 to 3.14.
LocationListenerVelocity	Double	The units are meters/second. The minimum value is zero.
LocationProjectionAngle	Double	Units are degrees and range is from zero to 180.
LocationSourceVelocity	Double	The units are meters/second. The minimum value is zero.
Medium	Integer	Medium for sound propagation: 0—Air, 1—Water.
ReferenceDistance	Double	Nominal distance for recording. The units are meters and minimum distance is zero.
ReverbAttenuation	Double	Reverberation model: mix level. The units are decibels.
RoomReflectivity	Double	Reverberation model: Bounce attenuation. The units are decibels.
RoomSize	Double	Reverberation model: Distance between walls. The units are meters and the minimum value is zero.
SourceMode	Integer	Type of filtering to apply. 0—Unfiltered, 1—Localized 2—Ambient, 3—Binaural.

Notes

The QT3DAudio class is for adding audio spatialization (special effects) to audio tracks. Add the QT3DAudio object to a QTSoundTrack with QTSoundTrack.Set3D method.

The properties of this class enable to set QuickTime parameters that are documented at the following URL:

http://developer.apple.com/techpubs/quicktime/qtdevdocs/APIREF/SOURCESIV/ssplocalizationdata.htm

See Also

EditableMovie, QTSoundTrack classes.

QTEffect Class

A QuickTime effect, generated by GetQTSMPTEEffect or GETQTCrossFadeEffect using a QTEffectSequence instance. See the example for the QTEffectSequence class.

The **QTEffect** class has no properties, methods, or events.

QTEffectSequence Class

Builds a QuickTime movie from two pictures and a QuickTime effect.

Use **QTEffectSequence** as a constructor to create a **QTEffectSequence** object.

Super Class

Object

Syntax

***result* = New QTEffectSequence(*effect*,*image1*,*image2*,*frames*)**

Part	Type	Description
result	**QTEffectSequence**	QTEffectSequence object, created from the values of the passed parameters.
effect	QTEffect	QT effect used in creating the sequence.
image1	Picture	Initial image in sequence.
image2	Picture	Final image in sequence.
frames	Integer	Number of frames in the sequence.

Properties

Property	Type	Description
Image	Picture	Image associated with a particular frame in the QT sequence.
Frame	Integer	Frame of QT sequence.

Notes

Use GetQTSMPTEEffect or GETQTCrossFadeEffect to obtain the value of *effect*.

Example

This example creates a QuickTime movie from two pictures, p1 and p2, using the cross-fade effect.

```
Dim sequence as QTEffectSequence
Dim theEffect as QTEffect
theEffect=GetQTCrossFadeEffect
sequence=New QTEffectSequence (theEffect,p1,p2,96)
```

See Also

EditableMovie, QTTrack, QTEffect, QTEffectSequence classes; GetQTCrossFadeEffect, GetQTSMPTEEffect functions.

QTGraphicsExporter Class

Used to export a graphic.

Super Class

Object

Properties

Name	Type	Description
CompressionQuality	Integer	0 = Minimum quality 256 = Low quality 512 = Normal quality 768 = High quality 1023 = Maximum quality 1024 = Lossless
DefaultExtension	String	Default file extension.
DesiredTargetDataSize	Integer	Target data size in bytes.
HasSettingsDialog	Boolean	True if the exporter object has a settings dialog box that can be displayed by RequestSettings.
OutputFileCreator	String	Mac File Creator.
OutputFileType	String	Mac File Type
RequestSettings	Boolean	Presents a dialog box. See Notes, below.
SettingsDescription	String	Description of settings.

Methods

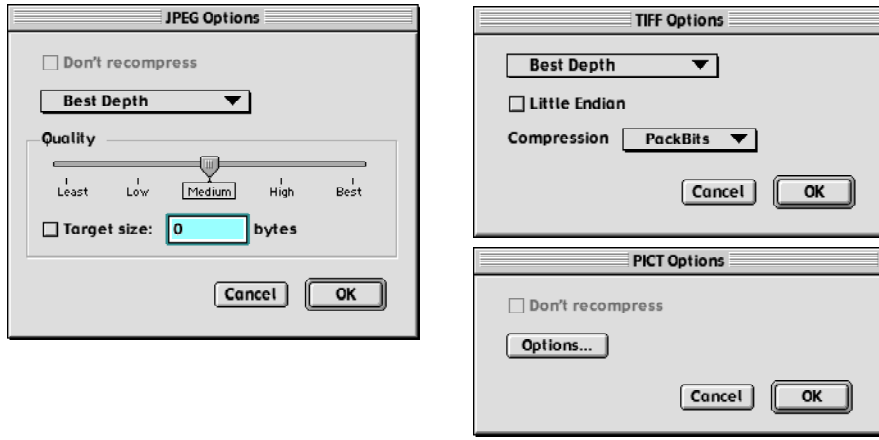
Name	Parameters	Description
SavePicture	Location as FolderItem, Image as Picture	Saves the image. Returns a Boolean.

Notes

The Request Settings dialog box has different options for different file types.

The following are some example RequestSettings dialog boxes:

Figure 1. Example RequestSettings dialog boxes.



Example

The following example exports a graphic, "lois", that has been dragged into the Project Window. You can either specify the exporter's properties (as shown) or display the Request Settings dialog box.

```
Dim f as FolderItem
Dim exporter as QTGraphicsExporter
Dim mybool as Boolean
f=GetFolderItem("HD:Graphic")
exporter=GetQTGraphicsExporter("JPEG")
//mybool=exporter.RequestSettings
exporter.DesiredTargetDatasize=36
exporter.CompressionQuality=512
exporter.OutputFileType="JPEG"
exporter.OutputFileCreator="ogle"
mybool=exporter.SavePicture(f,lois)
```

See Also

GetQTGraphicsExporter function.

QTSoundTrack Class

Enables you to add sound to EditableMovies.

Super Class Object

Properties

Name	Type	Description
Balance	Integer	Gets or sets the left-right balance of the **QTSoundTrack**. The range is ± 128.
Bass	Integer	Gets or sets the bass level of the **QTSoundTrack**. The range is ± 255.
Compressor	String	Gets the compressor used by the **QTSoundTrack**. The returned value is a Mac OS API OSType. For example, "raw " or "MAC3". Use this property to compare QTSoundTracks to avoid combining different compression codecs on the same QTSoundTrack when inserting or appending.
SampleRate	Integer	Use to get the sample rate of the **QTSoundTrack**. Use this property to compare QTSoundTracks to avoid combining different sample rates on the same QTSoundTrack when inserting or appending.
SampleSize	Integer	Use to get the sample size (bit depth) of the **QTSoundTrack**. Use this property to compare QTSoundTracks to avoid combining different sample sizes on the same QTSoundTrack when inserting or appending.
Stereo	Boolean	Use to determine whether the **QTSoundTrack** is in stereo. True indicates Stereo. Use this property to compare QTSoundTracks.
Treble	Integer	Use to get or set the treble level of the **QTSoundTrack**. The range is ± 255.
Volume	Integer	Use to get or set the volume level of the **QTSoundTrack**. The range is 0 to 512.

Methods

Name	Parameters	Description
AppendSoundTrackSegment	SourceTrack As **QTSoundTrack**, SourcePosition As Double, SourceDuration As Double, CopyMedia As Boolean, ShowProgress As Boolean	Adds a segment from another **QTSoundTrack** to the end of the calling QTSoundTrack. *SourcePosition* (seconds) is the beginning of the segment in SourceTrack to be appended to the calling **QTSoundTrack**. *SourceDuration* (seconds) is the length of the segment in *SourceTrack* to be appended to the calling **QTSoundTrack**. If *CopyMedia* is True, the media in *SourceTrack* will be copied to the destination track; otherwise a reference to SourceTrack will be used. If *ShowProgress* is True and the operation is lengthy, QuickTime will display a modal dialog box with a progress indicator as the operation proceeds.
Get3D		Obtains a reference to the QT3DAudio object in the movie. Returns the QT3DAudio object.
InsertEmptySoundTrackSegment	ThePosition As Double, TheDuration As Double	Inserts blank space in a **QTSoundTrack** that already has media on it. The parameters *ThePosition* and *TheDuration* specify in seconds the starting point and duration of the blank segment to be inserted. This method has no effect on a **QTSoundTrack** that has no media on it.

Name	Parameters	Description
InsertSoundTrackSegment	SourceTrack As **QTSoundTrack**, SourcePosition As Double, SourceDuration As Double, destinationPosition As Double, copyMedia As Boolean, ShowProgress As Boolean	Adds a segment from another **QTSoundTrack** to the calling QTSoundTrack. SourcePosition is the starting point (seconds) in SourceTrack of the segment to be inserted; SourceDuration is the length (seconds) of the segment; DestinationPosition is the starting point (seconds) in the calling QTSoundTrack where *SourceTrack* will be inserted. If *CopyMedia* is True, the media in *SourceTrack* will be copied to the destination track; otherwise a reference to SourceTrack will be used. If *ShowProgress* is True and the operation is lengthy, QuickTime will display a modal dialog box with a progress indicator as the operation proceeds.
Set3D	3DAudio as QT3DAudio	Sets audio localization of a **QTsoundtrack**.

QTTrack Class

A QuickTime track, created from an EditableMovie.

Super Class Object

Properties

Name	Type	Description
Duration	Double	Duration of track in seconds.
Enabled	Boolean	True if Enabled. A disabled track will not be displayed (if video) or heard (if audio).
MediaType	String	Four-character media type (soun, vide).
TimeDuration	Integer	Duration of the movie, in TimeScale units.
TimeScale	Integer	Defines the timescale of the track, in 1/TimeScale units per second.
TrackID	Integer	ID of track.

Methods

Name	Parameters	Description
DataSize	StartTime as Integer DurationTime as Integer	Returns Integer, number of bytes in segment specified by *StartTime* and *DurationTime*.
Delete		Deletes the **QTTrack**
DeleteTrackSegment	StartTime as Double, Duration as Double	Deletes a segment from a **QTTrack**. *StartTime* is the beginning of the segment to be deleted (seconds) and *Duration* is the length (seconds) of the segment.
FindFirstSample		Finds first sample (frame in the case of a video track).
FindNextSample		Finds next sample (frame for video track).
FindFirstKeyFrame		Finds first key frame (base frame from which differences are calculated)
FindNextKeyFrame		Finds next key frame.

Notes

The *TimeScale* parameter defines the number of time units that pass each second. For example, a TimeScale value of 60 sets the timescale in sixtieths of a second. The *TimeDuration* parameter is expressed in TimeScale units.

See Also

EditableMovie, QTVideoTrack classes.

QTUserData Class

Used to add and retrieve QuickTime user data, such as copyright notice, credits, and miscellaneous information.

Super Class

Object

Methods

Name	Parameters	Description
AddUserData	udType as String Value as String	Adds a user data item of type *udType* to that user data list.
GetUserData	udType as String Index as Integer ByRef Value as String	Gets user data of type *udType* at position given by Index. After successful call, the user data is stored in *Value*. Returns a Boolean.
GetUserDataText	udType as String Index as Integer ByRef Value as String Region as Integer	Gets user data of type *udType* for *Region*. After successful call, the user data is stored in *Value*. Returns a Boolean.
SetUserDataText	udType as String Index as Integer Value as String Region as Integer	Adds user data item in position given by *Index* of type *udType* for *Region*.
UserDataCount	udType as String	Number of items in *udType*. Returns an Integer

Notes

A QuickTime movie can contain user data lists, denoted by the *udType* parameter. UserData contains additional information about the movie, including copyright information, etc. The QTUserData class is used to manage the user data lists. UserData is a property of EditableMovie.

The udType parameter is a four character code for the type of user data. Apple's currently-defined user data types are shown in the following table:

udType	Description
©nam	Movie's name.
©cpy	Copyright statement.
©day	Date the movie was created.
©dir	Name of the movie's director.
©ed1 to ©ed9	Edit dates and descriptions.
©fmt	Indication of movie format.
©inf	Information about the movie.
©prd	Name of movie's producer.
©prf	Names of performers.

udType	Description
©req	Special hardware and software requirements.
©src	Credits for those who provided movie source content.
©wrt	Name of movie's writer.

Apple has reserved all lowercase data type values; however, you can create user data type values using uppercase letters. Apple recommends using type values beginning with the © character to specify user data items that store text data.

It is possible to create more than one user data item in a user data list. Therefore, each user data item is referenced by the *Index* parameter, where the first item is numbered 1. Please see the example for multiple items within a data type.

You may also provide alternate text for a given user data item, for example, to support multiple languages. The alternate versions are indicated by the value of the *Region* parameter. It represents a region code. The example shows English and French titles for a movie using the Region parameter.

Examples

The following code adds several performers and a director to the movie and then displays the names of the performers in a ListBox and the director in an EditField. It also adds English and French titles to the movie.

```
Dim mybool as Boolean
Dim myval as String
Dim i, myint as Integer
Dim f as FolderItem
Dim m as EditableMovie
f=GetFolderItem("FrmAmngTheDead.Mov")
If f <> Nil then
 m=f.OpenEditableMovie
 //Performers
 m.UserData.AddUserData("©prf","Kim Novak")
 m.UserData.AddUserData("©prf","James Stewart")
 m.UserData.AddUserData("©prf","Barbara Bel Geddes")
 m.UserData.AddUserData("©prf","Tom Helmore")
 //number of performers
 MyInt=m.UserData.UserDataCount("©prf")
 //the director...
 m.UserData.AddUserData("©dir","Alfred Hitchcock")
 //English and French titles
 m.UserData.SetUserDataText("©nam",1,"Vertigo",1)
 m.UserData.SetUserDataText("©nam",1,"D'Entre les Morts",2)
 //display Director's name
 mybool=m.UserData.GetUserData("©dir",1,myval)
 EditField1.text=myval
 //display cast members in ListBox
 For i=1 to MyInt
  mybool=m.UserData.GetUserData("©prf",i,myval)
 ListBox1.AddRow(myval)
 Next
end if
```

See Also

EditableMovie class

QTVideoTrack Class

A QuickTime video track.

Super Class QTTrack

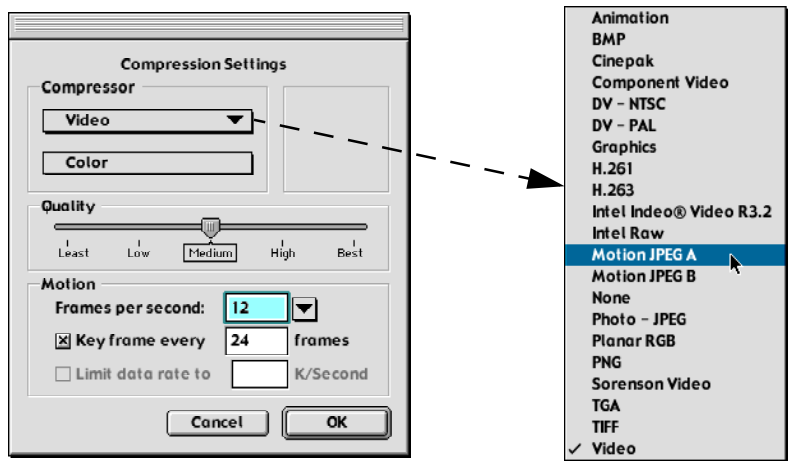
Methods

Name	Parameters	Description
AppendPicture	Pic as Picture	Picture to be appended to video track. Used to build a movie frame-by-frame.
AppendVideoTrackSegment	SourceTrack As **QTVideoTrack**, SourcePosition As Double, SourceDuration As Double, CopyMedia As Boolean, ShowProgress As Boolean	Appends a segment from *SourceTrack* to the calling **QTVideoTrack** *SourcePosition* is the starting point or the segment (seconds) in *SourceTrack* and *SourceDuration* is the length (seconds) of the segment. If *CopyMedia* is True, the media in *SourceTrack* will be copied to the destination track; otherwise a reference to SourceTrack will be used. If *ShowProgress* is True and the operation is lengthy, QuickTime will display a modal dialog box with a progress indicator as the operation proceeds.
InsertVideoTrackSegment	SourceTrack As **QTVideoTrack**, SourcePosition As Double, SourceDuration As Double, DestinationPosition As Double, CopyMedia As Boolean, ShowProgress As Boolean	Inserts a segment from SourceTrack to the calling QTVideoTrack at DestinationPosition. *SourcePosition* is the starting point or the segment (seconds) in *SourceTrack* and *SourceDuration* is the length (seconds) of the segment. *DestinationPosition* is the position in the calling QTVideoTrack at which the segment will be inserted. If *CopyMedia* is True, the media in *SourceTrack* will be copied to the destination track; otherwise a reference to SourceTrack will be used. If *ShowProgress* is True and the operation is lengthy, QuickTime will display a modal dialog box with a progress indicator as the operation proceeds.

Name	Parameters	Description
SelectCompressionSettings		Brings up the standard QT compression settings dialog box. Returns a Boolean that is True if the user clicked OK.

Notes

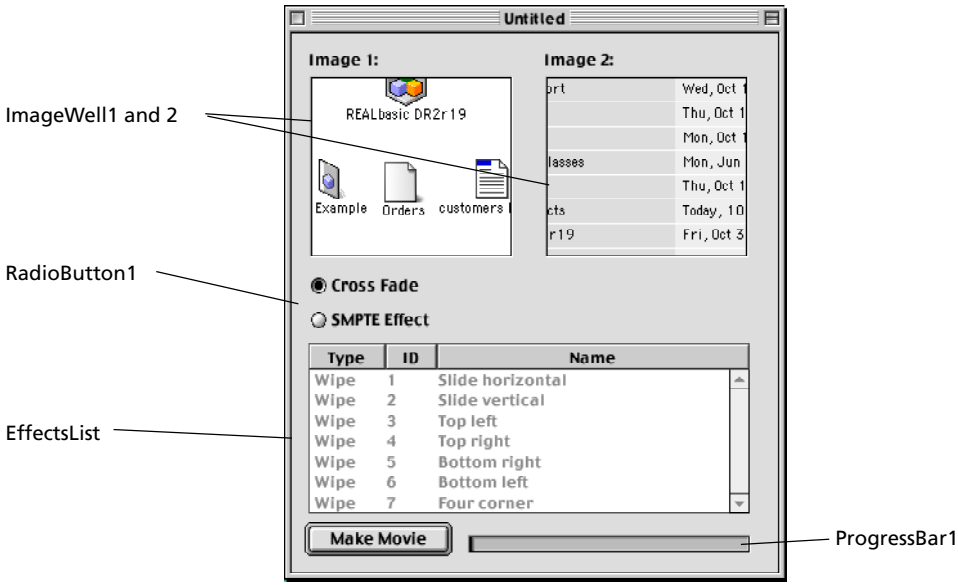
The SelectCompressionSettings method displays the following dialog box.



Example

The following example creates a QuickTime movie from two images and a user-selected video effect.

The interface for this example is as follows:



```
Dim sequence as QTEffectSequence
Dim i,effectID as Integer
Dim theEffect as QTEffect
Dim track as QTVideoTrack
Dim m as EditableMovie
Dim f as FolderItem
Dim mybool as Boolean
f=GetFolderItem("Movie")
m=f.CreateMovie //create null movie
If radiobutton1.value then //get user's chosen effect
 theEffect=GetQTCrossFadeEffect
else
 theEffect=GetQTSMPTEeffect(val(EffectsList.cell(EffectsList.ListIndex,1)))
end if
sequence=New QTEffectSequence(theEffect,ImageWell1.Image,
ImageWell2.Image,96)
  track=m.NewVideoTrack(ImageWell1.Image.width,
  ImageWell1.Image.height, 32)
mybool=track.SelectCompressionSettings
For i=1 to 96
 sequence.frame=i
 track.appendpicture sequence.image
 progressBar1.value=i
Next
f.launch //open movieplayer
```

See Also FolderItem, EditableMovie, QTTrack, QTEffect, QTEffectSequence classes; GetCrossFadeEffect, GetQTSMPTEEEffect functions.

Quaternion Class

A 3D object for representing orientation or rotation within an RB3DSpace world.

Properties

Name	Type	Description
W	Double	The W component of the Quaternion.
X	Double	The X component of the Quaternion.
Y	Double	The Y component of the Quaternion.
Z	Double	The Z component of the Quaternion.

Methods

Name	Parameters	Description
Inverse		Returns the opposite of the Quaternion. Returns a Quaternion.
Invert		Reverses the Quaternion to represent the opposite direction.
MultiplyBy	quat as Quaternion	Multiplies this Quaternion by another Quaternion, updating this Quaternion. Same as the Time method, except that it updates this Quaternion rather than returning the result as a Quaternion.
Normalize		Scales Quaternion to a standard size.
SetBetween	quat1 as Quaternion, quat2 as Quaternion, position as Double	Sets this Quaternion to a combination of *quat1* and *quat2*.
SetRotateAboutAxis	axisX as Double, axisY as Double, axisZ as Double, radians as Double	x, y, and z coordinates and the object's angle (radians) Used to rotate an object about the axis at the specified angle.
Times	quat as Quaternion	Multiplies *quat* by another Quaternion and returns the product. Returns a Quaternion.
Transform	vector as Vector3D	Returns the result of rotating *vector* by this Quaternion. Returns a Vector3D.

Notes

A **Quaternion** is a compact, efficient way to represent orientation or rotation in three-dimensional space. They do the same job for orientation and rotation that vectors do for position and movement. Any possible rotation can be represented as an angle turned around a certain axis. A **Quaternion** is essentially an axis plus an angle, although the internal representation is slightly different from this for mathematical reasons.

Just as a vector can be used to represent either position or displacement, a **Quaternion** can be used to represent either orientation or rotation. To represent absolute orientation, we simply consider the quaternion as a rotation relative to a "null" or standard orientation.

The neat thing about quaternions is that they can be combined in a very straightforward manner. When you multiply quaternion A by quaternion B, the result is a quaternion which we'll call C. Rotation C puts the object in exactly the same orientation as you'd get by doing rotation A followed by rotation B. Multiplying quaternions is exactly the same as composing rotations.

To make a quaternion represent a particular rotation, use the SetRotateAboutAxis method, giving it the axis and the angle (in radians) you want to rotate. For example, to make a quaternion that rotates 90 degrees around the X axis, you'd use:

```
q.SetRotateAboutAxis 1,0,0, Pi/2
```

To rotate a vector, use the Transform method of a **Quaternion**. Give it a vector, and the result will be a new vector, rotated about the origin (0,0,0).

To interpolate between two Quaternions: this is used when you think of the quaternions as representing orientation. You have q1 representing the object in one orientation and q2 representing it in another. You want to animate the object smoothly rotating between q1 and q2. To do this, set the object's orientation to something in between q1 and q2 by using the SetBetween method. The final parameter is the position between these two, with 0.0 meaning q1 and 1.0 meaning q2. 0.5 would be a position exactly halfway in between.

One common application is to update an object's orientation by some rotation. For example, suppose Object3D didn't have a built-in Yaw method. You could do the same thing yourself as follows:

```
Dim yawQuat as Quaternion
yawQuat = New Quaternion
yawQuat.SetRotateAboutAxis 0,1,0, yawAngle  // "yaw" -rotate about Y axis
obj.orientation = yawQuat.Times(obj.orientation)
```

Note that when multiplying quaternions, order does matter. A.Times(B) is not the same as B.Times(A); this is just the nature of rotations in 3D space.

Finally, note that quaternions also have a MultiplyBy function; Q1 = Q1.Times(Q2) can be written more efficiently as Q1.MultiplyBy Q2. But that doesn't help in the example above, because we need Q2 = Q1.Times(Q2) in order to get the correct ordering. We want to start with the current orientation of the object, so it must appear on the right.

Interpolation

Use the SetBetween method, giving it the two quaternions you want to interpolate between, and the interpolation factor: 0.0 to exactly match the first quaternion, 1.0 to exactly match the second, and something else to assume some position in between:

```
Q.SetBetween Q1, Q2, 0.25   // set Q to 25% of Q1 plus 75% of Q2
```

This is an amazingly useful technique. It lets you produce smooth animation between the orientation that you have and an orientation you want to have. For example, if you

have two "keyframe" (pre-defined or pre-computed) positions and orientations the object is supposed to interpolate between, in 11 steps, you could do it this way:

```
For i = 0 to 10
 obj.position.x = position1.x * i/10 + position2.x * (1 - i/10)
 obj.position.y = position1.y * i/10 + position2.y * (1 - i/10)
 obj.position.z = position1.z * i/10 + position2.z * (1 - i/10)
 obj.orientation.SetBetween orientation1, orientation2, i/10
 Rb3DSpace1.Update
next
```

See Also Vector3D, Object3D, Group3D, classes; Rb3DSpace control.

Quit Method

Quits the application.

Syntax **Quit**

Notes If any windows are open, calling the **Quit** method will call each window's CancelClose event handler. If the CancelClose event handler returns False (the default action) then the window's Close event handler will be executed. If any window's CancelClose event handler returns True, REALbasic will stop sending CancelClose or Close events and the application will not quit.

If no windows are open or no CancelClose event handlers return True, and a subclass exists based on the Application object, its Close event will be executed.

Example This example quits the application.

```
Quit
```

QuitMenuItem Class

The Quit menu item is derived from the **QuitMenuItem** class.

Super Class

Object

Properties

Name	Type	Description
BalloonHelp	Message as String	Message that appears when Balloon Help is on and the user holds the mouse over the menu item while the item is enabled.
Bold	Boolean	Indicates if the menu item should appear in bold.
Checked	Boolean	Indicates whether or not the menu item is checked.
CommandKey	String	The Macintosh keyboard shortcut for the menu item. Keyboard shortcuts for Windows are set using Constants. See the section "Using Constants to Add Keyboard Shortcuts to Menus and Menu Items" in the *User's Guide*.
DisabledBalloonHelp	Message as String	Message that appears when Balloon Help is on and the user holds the mouse over the menu item while the item is disabled.
Enabled	Boolean	Indicates whether or not the menu item is enabled. Menu items based on this class are enabled by default. You can disable a menu item either by assigning the Enabled property a value of False.
Italic	Boolean	Indicates if the menu item should appear in italic.
Name	String	The name of the menu item.
Submenu	Boolean	Indicates that the menu item is a submenu.
Tag	String	A 'hidden' text string associated with the menu item. The tag is accessible via code when the user chooses the menu item but, unlike the Text property, is not displayed in the menu.
Text	String	The text of the menu item.
Underline	Boolean	Indicates if the menu item should appear underlined.
Visible	Boolean	Indicates whether or not the menu item is visible.

Methods

Name	Parameters	Description
Close		Removes dynamically created menu items.
Enable		Sets the Enabled property of the menu item to True. Call Enable only within an EnableMenuItems event handler.

Notes

The Quit menu item that is added by default to all applications is an instance of the **QuitMenuItem** class. It differs from menu items derived from the MenuItem class in that it is enabled by default and automatically calls the Quit method to quit the application. Thus, there is no need to enable the menu item via code or write its handler.

Examples

You normally have no need to create additional instances of the **QuitMenuItem** class, but you can use its properties to modify the default appearance or behaviour of the Quit menu item.

```
FileQuit.BalloonHelp="Exits the application"
```

See Also

EnableMenuItems function, New operator; MenuItem class.

RadioButton Control

The standard radio button control. If you enclose a group of radio button controls within a GroupBox control, REALbasic will automatically deselect the remaining radio buttons when the user clicks a radio button in the group.

Super Class

RectControl

Because this is a RectControl, see the RectControl for other properties and events that are common to all RectControl objects.

Properties

Name	Type	Description
Bold	Boolean	Applies the bold style to the button caption.
Caption	String	The button's text.
Italic	Boolean	Applies the italic style to the button caption.
TextFont	String	Name of the font used to display the button caption.
TextSize	Integer	Size of the font used to display the button caption.
Underline	Boolean	Applies the underline style to the button caption.
Value	Boolean	Value of the radio button when its owning window opens.

Events

Name		Description
Action		The button has been clicked.

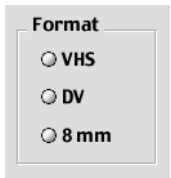
Name		Description
GotFocus		(Win32 only) The button has received the focus and has a selection marquee around its caption.
LostFocus		(Win32 only) The button has lost the focus and no longer has a selection marquee.
MouseDown	x as Integer, y as Integer	The mouse button was pressed inside the control's region at the location passed in to x,y. Return True if you are going to handle the MouseDown. The Action event will not execute and the state of the object will not change.
MouseUp	x as Integer, y as Integer	The mouse button was released inside the control's region at the location passed in to x,y. This event will not occur unless you return True in the MouseDown event. The return value is ignored.

Note

If the Caption property contains an ampersand character, it will display only if it is preceded by another ampersand character. This is done to make Macintosh and Windows applications behave consistently.

Example

The following interface uses a GroupBox and three RadioButtons:



The following code is used in each RadioButton's Action event handler:

```
If radiobutton1.value then
  EditField1.text="VHS"
end if
```

Similar code is used in the Action event handlers for RadioButtons 2 and 3.

See Also

GroupBox control; RectControl class.

Raise Statement

The Raise statement is used to raise an exception.

Syntax

Raise *expression*

Expression must evaluate to an object that is a subclass of RuntimeException — and will be the object that is passed to any exception handlers.

Example

```
Dim f as FolderItem
f.delete
Exception err as NilObjectException
Raise err
```

See Also

Function statement, RuntimeException class.

Random Class

Use to generate random numbers.

Super Class

Object

Properties

Name	Type	Description
Seed	Integer	Gets or sets the seed used by the random number generator.

Methods

Name	Parameters	Description
InRange	minR as Integer maxR as Integer	Returns a random number as an Integer in the range from *minR* to *maxR*.
LessThan	Range as Integer	Returns a random number as an Integer in the range greater than or equal to 0 and less than *Range*-1
Number		Returns a random number as a Double in the range greater than or equal to 0 and less than 1.

Notes

The **Random** class supersedes the Rnd function. Please recode to use the **Random** class.

Example

The following example generates a random integer in the range from 0 to 1,000.

```
Dim r as Random
r=New Random
Statictext1.Text=Str(r.InRange(0,1000))
```

See Also

Rnd function.

RB3DSpace Control 

Used to view and manipulate objects in 3D space.

Properties

Name	Type	Description
AmbientColor	Color	The color of the built-in ambient light.
AmbientLight	Integer	Brightness of ambient light. Range is from 0 to 100.
Background	Object3D	Background object or grouped object (Group3D).
Camera	Object3D	Camera position and orientation.
DebugCube	Boolean	If True, draws an x, y, z set of axes in the background of the RB3DSpace help orient you. The debug cube appears only in the IDE. Not available in REALbasic version 5.0.
FieldOfView	Integer	Angle of view (degrees).
FloodColor	Color	The color of the light from the built-in floodlight. The default color is white.
FloodDirection	Vector3D	Specifies the direction of the built-in floodlight. If Nil, you get a point light that radiates in all directions.
FloodLight	Integer	Brightness of floodlight. Range is from 0 to 100.
FogStart	Integer	If FogVisible is True, indicates how far away from the camera the fog should begin. The fog gets thicker from that point (or Hither), whichever is farther away, until the Yon distance, where it is fully opaque. The default is zero. The fog color is the same as the SkyColor.
FogVisible	Boolean	If True, fog is drawn according to the FogStart parameter. Default is False.
Hither	Integer	Near clipping plane. Units are arbitrary, but are the same for Yon and FogStart.
LightCount	Integer	The number of lights in the scene.
Objects	Group3D	Objects to be rendered.
SkyColor	Color	Background color.

Name	Type	Description
WireFrame	Boolean	If True, render as wireframe.
ViewHandle	Integer	Allows you to access the TQ3ViewObject associated with the control, for use with Declares to Quesa/QD3D. The object returned is not locked, so you should not dispose of it nor attempt to use it after the control is closed.
Yon	Integer	Far clipping plane. Units are arbitrary, but are the same as Hither and FogStart.

Methods

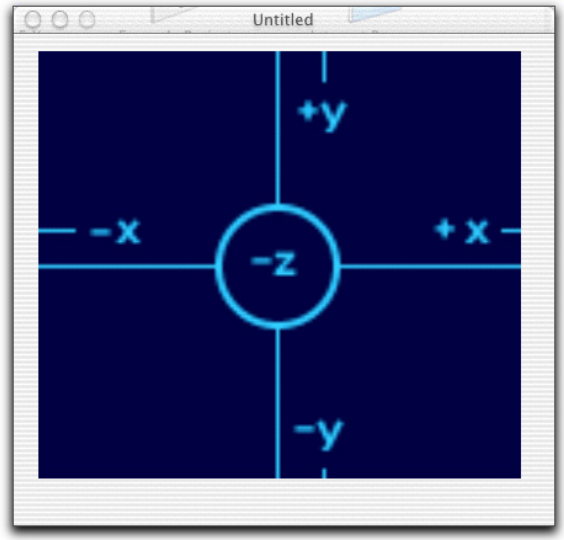
Name	Parameters	Description
AddLight	Light as Light3D	Adds a light to the scene.
FindObject	x as Integer y as Integer	Finds object at x, y (pixels) within the RB3DSpace. The coordinates. Returns Nil if the point contains no object. Returns the object drawn at that point as an Object3D.
FindPoint	x as Integer, y as Integer	Finds point at x, y (pixels) within the RB3DSpace. Returns Nil if the x, y, contains no point. Returns returns the 3D world coordinates of the part of the scene drawn at that point as a Vector3D.
GetPicture	[Width as Integer], [Height as Integer], [Depth as Integer]	Renders the current scene into a newly created Picture and returns it. If the parameters are omitted the control's current width and height and a depth of 32 are used.
Light	Index as Integer	Gets or sets a light.
RemoveLight	Index as Integer OR Light as Light3D	Removes the specified light from the scene. You can indicate the light to be removed either by its index or by reference to it.
Update		Redraws the 3D space without erasing or triggering a refresh.

Notes

Requirements The 3D classes in REALbasic require either Quesa or QuickDraw 3D to work. The latter is a library by Apple Computer and part of QuickTime, while Quesa is an open-source project that does pretty much the same thing. Which one you use may depend on the operating system you're running.

Under Mac OS "classic," QuickDraw 3D is installed by default, so you don't need to install any additional libraries. You can instead install Quesa Classic if you prefer.

Under OS X or Carbon, you need to install Quesa (and OpenGL). If Quesa is installed, you will see the x, y and z axes when you set the DebugCube property to True, as shown below.



Under Windows, you have a choice. QuickDraw 3D is part of QuickTime 5 for Windows, so you can use that or Quesa (and OpenGL). The Rb3DSpace control is written assuming Quesa, and since Windows locates libraries based on their file names, to use QuickDraw 3D you'll need to rename QD3D.DLL to QUESA.DLL. (This is not recommended; Quesa gives better performance under Windows.)

If the required libraries are not installed if the Objects property of any Rb3DSpace control will be Nil. If the proper libraries are available, this property is automatically initialized to a Group3D (i.e., it's non-nil). So a check like the following (in the Open event of the Rb3DSpace control) is a good idea:

```
If Me.objects = Nil then
 MsgBox "You need to install QuickDraw 3D or Quesa!"
 Return
End if
```

Note that this works only under Mac OS. Under Windows, if the QUESA.DLL library can't be found, your program will not even launch. This is just a side-effect of the way shared libraries work under Windows.

Background

The RB3DSpace control is where 3D renderings are actually done. The axes of the 3D space are displayed when you set the DebugCube property to True, as shown below. The Z axis is the "depth" dimension and the "size" of this axis is controlled by the Hither and Yon properties. The Hither and Yon properties define the near and far clipping planes (the Z axis); you want to keep Hither and Yon as close together as possible in order to make best use of the Z-buffer of your hardware. FieldOfView is in degrees, and specifies how wide is the angle that the camera sees; make this smaller to zoom in, or larger to zoom out.

Note that the camera is defined as an object; it doesn't need any geometry, but this lets you move the camera exactly as you would move any other object. The background is defined as an Object3D, but you could assign a Group3D to it instead if you want.

All other objects in the scene are stored in the Objects property, which is a Group3D object so that you can easily add things to it (or iterate over objects in the scene, etc.). If you need to find the object or point in 3D space which corresponds to a given pixel position within the view, use FindObject and FindPoint. (These will return Nil if the point clicked contains no object.)

Specifying a Location in 3D Space

Any location in a 3D space can be specified the three coordinates: X, Y, and Z. If you imagine the world as one infinite, flat plane, you would pick some arbitrary point as the "origin." Then you can specify any location by a distance East/West from the origin, a distance North/South, and a height/depth.

The arbitrary center of the virtual universe is called the "origin." By convention, the three coordinates in 3D graphics are called X, Y, and Z and the origin is the point 0,0,0. The X-axis is the line through the origin formed by assuming Y=0 and Z=0; similarly, the XY plane is the infinite plane through the origin where Z=0.

The standard interpretation of the axes is that the XZ plane is the ground (or parallel to the ground), and the Y coordinate is height. In REALbasic, the three axes in REALbasic are arranged such that if you're looking in the -Z direction, with +Y pointing up, then +X is to the right. (This is the default orientation of the camera, too.) So, if you're looking in this direction, than an increase in the X coordinate of an object's position will cause it to move to the right from your point of view. An increase in Z will cause it to move closer to you, and a decrease in Z will cause it to move further away.

The FieldOfView Property

FieldOfView is the angle, in degrees, that's shown in the area of the RB3DSpace control. A normal human has a field of view of about 180 degrees, but the monitor itself is only on the order of 10 degrees across for a typical viewing distance. If you try to compress a large field of view down to a display only 10 degrees across, you'll cause a "fisheye" effect which looks very unnatural. On the other hand, if you use only a 10 degree field of view in a typical 3D environment, users will feel like they've got tunnel

vision and will constantly be panning the camera around. The FieldOfView you choose is a balance between these two conflicting constraints: You want to see a useful amount of the scene, but you want it drawn as realistically as possible.

A typical FieldOfView value for many 3D games is about 40 to 50 degrees. However, you can provide a "zoom" (telescopic vision) option simply by reducing the FieldOfView temporarily — a smaller FieldOfView means that you're displaying a smaller part of the world in the same area on screen, hence everything in that part looks bigger.

Finally, the telescopic quality of a small FieldOfView also allows you to reduce (and nearly eliminate) perspective when you need to. Put the camera very far away from the scene and use a small FieldOfView; this is equivalent to looking at the ground from orbit, and under those conditions, there is almost no foreshortening.

The Hither and Yon Properties

These are standard 3D graphics jargon for the "near-far" (Z) axis. Nothing closer to the camera than the Hither distance gets drawn; and if an object happens to be halfway straddling Hither, it will simply be cut in half. Everything closer than Hither is clipped out of the scene (thus, the "near clipping plane"). Similarly, nothing farther from the camera than *Yon* is drawn.

If there were nothing more to it, you would simply set Hither to zero and Yon to some very large number so that things aren't clipped. But in reality, that's a very bad idea. The reason is that video cards have a limited amount of resolution in the "depth buffer" that's used to keep track of what's in front of what. The ratio Yon/Hither must be evenly divided by the number of bits in the card's depth buffer. If that ratio is very large, then the depth resolution will be inadequate and you'll see all sorts of visual artifacts — objects mixed together in odd ways, further-away objects being drawn in front of closer ones, etc. So it's very important to the quality of your rendering that you keep the Yon/Hither ratio as small as possible. Increase Hither and decrease Yon as much as you can in your application.

Orienting an object

There are an infinite number of orientations that all point towards a certain point of interest (or "POI"). Prove this to yourself by taking (or imagining) an actual camera with a view-finder. Look through the view-finder at some object, say a tree down the street. Now, without losing sight of the tree, rotate the camera around the axis between it and the POI. There are an infinite number of different orientations you could stop the camera in, even though they all look at the tree.

The first thing one must do is define some additional constraint on the orientation. A common one is to keep the object as "upright" as possible. There are still many ways to interpret this, but here's one solution:

```
Sub PointObjectAt(obj As Object3D, POI As Vector3D)
  // Make object 'obj' point towards the point of interest 'POI',
  // by first doing a yaw (from the default orientation), and
  // then a pitch. The object's previous orientation is not
  // involved.

 Dim yawAngle As Double
 Dim pitchAngle As Double
 Dim v1, v2 As Vector3D

  // Find the yaw angle needed to get in the right general direction
  // (i.e., ignoring Y, get it facing the right XZ direction).
  // The Atan2 function provides a very easy way to find this.
 yawAngle = Atan2(POI.x-obj.position.x, POI.z-obj.position.z)

  // Set the object's orientation to a yaw by that amount.
 obj.orientation.SetRotateAboutAxis 0,1,0, yawAngle

  // Now, find the pitch angle needed to account for Y.
  // We use a dot product to get the angle between a vector in
  // the direction the object's facing now, and a vector that
  // points towards the POI.
 v1 = New Vector3D
 v1.z = 1.0
 v1 = obj.orientation.Transform(v1)
 v2 = New Vector3D
 v2.x = POI.x - obj.position.x
 v2.y = POI.y - obj.position.y
 v2.z = POI.z - obj.position.z
 v2.Normalize
 pitchAngle = Acos( Min(Max(v1.Dot(v2),-1.0),1.0) )
 If v2.y > 0 then
  pitchAngle = -pitchAngle
 end if

  // Apply that pitch.
 obj.Pitch pitchAngle

End Sub
```

Orbiting an Object around a POI

This is another task that can be interpreted in many ways. Start by figuring out exactly what want to do. If the object in "orbit" is a camera that you want to move around a Point Of Interest (POI), there are several reasonable options:

- Move the camera on the surface of an imaginary, Y-axis-aligned cylinder which surrounds the object. Left-right movements will move the camera laterally around the cylinder, and up-down movements will move the camera up and down the side of the cylinder. Note that this means that the camera gets further away from the POI when you move up or down. But it also means that the left-right/up-down directions are always very well defined and obvious.
- Move the camera on the surface of an imaginary sphere around the object. The camera object will have a certain direction it's facing (towards the center), and left-right/up-down will move the camera on the surface of the sphere relative to that direction. This keeps a constant distance from the POI, but the controls can get confusing since the moving object can easily end up upside down or sideways.
- Move the camera on the surface of an imaginary sphere around the object. In this case, let left-right control the longitudinal position of the camera, and up-down control the latitude. This keeps the camera a constant distance from the POI, and is fairly easy to control since (like the cylindrical case) the left-right/up-down directions are always well defined.

  In all these cases, the position of the object can be described by two parameters. Left-right controls increase or decrease one of the parameters, and up-down controls adjust the other. The only difference among them is how these parameters are converted into 3D coordinates (and, therefore, what the parameters mean). In all cases, you could add a third parameter, radius, to change the size of the orbit.

  Here is the code for the first option:

```
Function OrbitCylinder(POI As Vector3D, radius As Double, angle As Double,
altitude As Double) As Vector3D
 // Compute a position on a virtual Y-aligned cylinder, centered on POI,
 // with the given radius, angle, and altitude.
 Dim out As Vector3D
 out = New Vector3D
 out.x = POI.x + radius * Cos(angle)
 out.z = POI.z + radius * Sin(angle)
 out.y = POI.y + altitude
 Return out
End Function
```

Example

This example shows how to load a 3DMF file into an **RB3DSpace**. This code is placed in the Open event. Note that "3DMF" should be declared in the File Types dialog, with Type code "3DMF".

```
Dim f As FolderItem
Dim modelCode As String
Dim obj As Object3D

f = GetFolderItem("Penguin.3DMF")
If f <> Nil then
 // get the 3DMF code for the model
 modelCode = f.OpenAsTextFile.ReadAll

 // create an object
 obj = New Object3D

 // give the object a shape specified by the 3DMF
 obj.AddShapeFromString modelCode

 // add the object to this Rb3DSpace
 Me.objects.Append obj
end if

// move the camera so that we can see the object
Me.camera.position.Z = 500
```

The last line moves the camera away from the object so it is visible. By default the camera is at location 0,0,0 (a.k.a., the "origin" in the 3D space). By default, objects are loaded at the same place. This means that the camera is inside your object, which is usually not where you want it to be. Fix this by moving the camera away from the origin. How far you need to move depends on how big your object is.

See Also

Object3D, Group3D, Vector3D, Quaternion, Bounds3D, Light3D classes.

RBScript Control 

Used to execute REALbasic code within a running (compiled) application.

Super Class

Object.

Properties

Name	Type	Description
Context	Object	Object (e.g., window or class) that is made available to the RBScript. For example, you can create a custom class and assign it to the Context property. The script would then have access to the methods and properties of this class. The methods become global methods and the properties become global properties.
CurrentLineNumber	Integer	The line the interpreter is executing. In version 4.5 (and above), the interpreter is not being used, so this will always be zero.
EncodingFont	String	Useful only for double-byte character systems. This font is used to determine what script system is used to interpret strings in functions such as Len and Mid. For example, to force interpretation of strings as Japanese script, set this property to "Osaka". If left blank, the system script is used.
Source	String	The source code the interpreter will run.
State	Integer	State of the script interpreter: 0=Ready 1=Running 2=Complete 3=Aborted.

Methods

Name	Parameters	Description
PartialRun	millisec as Integer	Runs the code *Source* until it is done or *millisec* has elapsed.
Precompile()		Runs the parser immediately instead of waiting until the next call to Run.
Reset()		Rewinds the interpreter to zero.
Run()		Runs the code *Source* until it is done.

Events

Name	Parameters	Description
CompilerError	line as Integer, errorNumber as Integer, errorMsg as String	A compiler error has occurred. This could be caused by a syntax error in the code. See the error numbers in Notes, below.

Name	Parameters	Description
Input	prompt as String	The script requests input from the end user. Returns a String.
Print	msg as String	The interpreter is returning the results of the script in *msg*.
RuntimeError	line as Integer, error as RuntimeException	A runtime error has occurred.

Notes

The RBScript language is an implementation of the REALbasic language that allows end users to write and execute REALbasic code within a compiled application. Scripts are compiled into machine language instead of running through the interpreter, as in versions prior to 4.5.

Since RBScript is a class, you use it by creating an instance of this class either via code or by dragging the RBScript tool from the Controls palette to a window. The easiest way to use RBScript is to assign the code to the Source property of the RBScript object and call the Run method.

To provide information to an RBScript while it's running, use the Input function. This function calls the Input event of the RBScript object where you can return the information you wish returned by the Input function in your RBScript code. In the following example, the results of the Input function are assigned to a variable:

```
Dim Years, Days as Integer
Years = Val(Input(" "))
Days = Years * 365
```

The Input function takes a String that can be used to provide a prompt in case you are going to provide a dialog box in which the user enters the information. Since the Input function returns a String and we want to store the value as an integer, the Val function is used to convert the string to an integer. In this case, if the number of years is going to be entered into an EditField called Editfield1, then the Input event of the RBScript object would look like this:

```
Function Input(prompt as String) as String
 Return Editfield1.text
End Function
```

When the Run method of the RBScript object is called, the code will be compiled and then executed. Since the Input function is called, the Input event of the RBScript object is executed and the contents of the Text property of the EditField is returned and assigned to the Years variable. Then the Days value is calculated.

Getting Information From RBScript

You obtain data from the RBScript using the Print method. This method takes a String and passes it to the Print event of the RBScript object. Here is the example modified to use the Print function:

```
Dim Years, Days as Integer
Years = Val(Input(" "))
Days = Years * 365
Print Str(days)
```

You access the value passed to the Print method through the Print event. For example, if you want to assign the value to the Text property of a StaticText object, the code for the Print event of the RBScript object would look like this:

```
Sub Print(msg as String)
 StaticText1.text = msg
End Sub
```

Handling Errors in Your Code

If an error occurs while RBScript is compiling your code, the CompilerError event of the RBScript object will be called and will be passed appropriate error information so you can then decide how to handle the error. If the error occurs while the code is running, the RuntimeError event of the RBScript object is called and is passed appropriate error information. You can then decide how to respond to the error.

Variables and Constants

All numeric and alphanumeric operators are supported:

+, -, *, /, \, Mod, <, =, >, <=, >=, <>.

Logical operators: And, Not, Or.

All forms of comments are supported: ', //, and REM.

Data Types

RBScript supports the following data types:

- Integer
- Single
- Double
- Boolean
- String
- Color
- Object
- Variant

Arrays can use any of these types.

Control Structures

All the control structures specified in this *Language Reference* are supported. This includes:

- Function... End Function
- Sub... End Sub
- For... Next
- Do... Loop
- If... Then... End If
- Select Case... End Select
- While…Wend
- Return statement

Classes

You can create a class using the Class…End Class structure. A new class can have properties, methods, and events.

```
Class NewClass
 Dim i as Integer //Property definitions..
 Dim c as Color
 Sub myMethod (x as Integer, y as Integer)
  //Method definition goes here
 End Sub
 //Class definition
End Class
```

A class can be subclassed from another class using the "Inherits" declaration.

Modules

You can create modules using the Module structure. For example:

```
Module Foo
 Dim bar As Integer
 Sub Baz ()
  bar = 42
 End Sub
End Module
```

Modules are similar to REALbasic modules. The differences are that properties are always protected and constants are not allowed. You can get the effect of a public property or constant by simply putting the Dim or Const statement outside of the module.

TypeCasting Typecasting is supported. Use the desired type's name as a function whose only argument is the object reference you want to cast. If it fails, it will trigger an IllegalExceptionError.

"Super" keyword The new "Super" keyword lets you call overridden methods without having to specify which class the method came from. This works for constructors as well. You can call overridden superclass methods using REALbasic's *classname.methodname()* syntax.

Inheritance Class inheritance is implemented with the "Inherits" declaration.

```
Class MyNewSubClass
 Inherits NewClass
 //continue coding here....
End Class
```

Events Classes can declare new events using a New Event declaration.

```
Class myClass
 New Event myEvent (myParameter as DataType) as DataType
End Class
```

Methods can handle such events using the Handles Event declaration. For example:

```
Class myClass
 New Event myEvent (myParameter as DataType) as DataType
End Class
Class mySubClass
 Inherits myClass
 Function myEvent (myParameter as DataType) Handles Event
 //continue with function here
 End Function
End Class
```

Interfaces RBScript now supports interfaces, declared by the Interface...End Interface structure. Interface methods are declared with a Sub or Function line just as they would be declared inside a class, but they have no contents and need no End Sub or End Function line:

```
Interface MovableItem
 Sub Move( x As Integer, y As Integer)
  Function LocationX() As Integer
  Function LocationY() As Integer
End Interface
```

NOTE: Function declarations with no parameters must have empty parentheses, as shown above.

A class may declare that it implements an interface with the Implements keyword:

```
Class Box
 Implements MovableItem
End Class
```

Classes can implement any number of interfaces, but they must provide all of the methods listed in the interface. Interfaces can be used in all the situations as in REALbasic.

Input and Output

Function	Comments
Input(Prompt as String)	Retrieves input from the user. This function triggers the Input event.
Print(String)	Sends output to the implied output device. This function triggers the Print event.

Functions

Standard library functions are supported as follows. These functions work as specified in this *Language Reference* unless comments indicate otherwise.

Function	Comment
Abs(Double) As Double	
Acos(Double) As Double	
Asc(String) As Integer	
AscB(String) As Integer	
Asin(Double) As Double	
Atan(Double) As Double	
Atan2(Double, Double) As Double	
BitwiseAnd(Integer, Integer) As Integer	
BitwiseOr(Integer, Integer) As Integer	
BitwiseXor(Integer, Integer) As Integer	
ByRef keyword	
ByVal keyword	
CDbl(String) As Double	Identical to Val().
Ceil(Double) As Double	
Chr(Double) As String	
ChrB(Double) As String	
CMY	
Color	
Const name = value	

Function	Comment
Cos(Double) As Double	
CountFields(String, String) As Integer	
CStr(Double) as String	
Dim	Allows you to declare multiple variables with different types rather than only multiple variables of the same type. Empty arrays can be created by specifying -1 elements.
Do...Loop	
Exit	
Exp(Double) As Double	
False	
Floor(Double) As Double	
For...Next	Allows floating point loop counters and step values.
Format(Double, String) As String	
Function	Functions can be called before they are defined.
Goto keyword	
Hex(Integer) As String	
HSV	
If...Then...Else	
InStr(Integer, String, String) As Integer	
InStrB(Integer, String, String) As Integer	
IsA operator	Can be used with Object; it will return True for any non-null object.
Left(String, Integer) As String	
LeftB(String, Integer) As String	
Len(String) As Integer	
LenB(String) As Integer	
Log(Double) As Double	
Lowercase(String) As String	
LTrim(String) As String	
Max(Double, Double) As Double	
Me	
Microseconds As Double	
Mid(String, Integer, Integer) As String	
MidB(String, Integer, Integer) As String	

Function	Comment
Min(Double, Double) As Double	
Nil	
NthField(String, String, Integer) As String	
Oct(Integer) As String	
Pow(Double, Double) As Double	
Redim	
Rem	
Replace(String, String, String) As String	
ReplaceB(String, String, String) As String	
ReplaceAll(String, String, String) As String	
ReplaceAllB(String, String, String) As String	
RGB	
Right(String, Integer) As String	
RightB(String, Integer) As String	
Rnd As Double	
Round(Double) As Double	
RTrim(String) As String	
Select Case	
Self	
Sin(Double) As Double	
Sqrt(Double) As Double	
Str(Double) As String	
StrComp(String, String, Integer) As Integer	
Sub	
Tan(Double) As Double	
Ticks as Integer	
Titlecase(String) As String	
Trim(String) As String	
True	
Ubound(array) As Integer	Supports second parameter for multi-dimensional arrays.
Uppercase(String) As String	
Val(String) As Double	Scientific format not recognized.
While...Wend	

Compiler error numbers returned in *errorNumber* are shown below:

Error Number	Description
1	Syntax does not make sense.
2	Type mismatch.
3	Select Case does not support that type of expression.
4	The compiler is not implemented (obsolete).
5	The parser's internal stack has overflowed.
6	Too many parameters for this function.
7	Not enough parameters for this function call.
8	Wrong number of parameters for this function call.
9	Parameters are incompatible with this function.
10	Assignment of an incompatible data type.
11	Undefined identifier.
12	Undefined operator.
13	Logic operations require Boolean operands.
14	Array bounds must be integers.
15	Can't call a non-function.
16	Can't get an element from something that isn't an array.
17	Not enough subscripts for this array's dimensions.
18	Too many subscripts for this array's dimensions.
19	Can't assign an entire array.
20	Can't use an entire array in an expression.
21	Can't pass an expression as a ByRef parameter.
22	Duplicate identifier.
23	The backend code generator failed.
24	Ambiguous call to overloaded method.
25	Multiple inheritance is not allowed.
26	Cannot create an instance of an interface.
27	Cannot implement a class as though it were an interface.
28	Cannot inherit from something that is not a class.
29	This class does not fully implement the specified interface.
30	Event handlers cannot live outside of a class.
31	It is not legal to ignore the result of a function call.
32	Can't use "Self" keyword outside of a class.
33	Can't use "Me" keyword outside of a class.
34	Can't return a value from a Sub.
35	An exception object required here.
36-39	Obsolete.
40	Destructors can't have parameters.
41	Can't use "Super" keyword outside of a class.
42	Can't use "Super" keyword in a class that has no parent.

RbScriptAlreadyRunningException error

Occurs if the user tries to modify an RBScript's source code while it is running or you try to change the Context object while the script is running. Don't do this.

Super Class RuntimeException.

See Also RbScript class, Exception Block.

RBVersion Constant

Reports the major and minor version number of REALbasic.

Syntax ***result*=RBVersion**

Part	Type	Description
result	Double	Constant indicating major and minor version number.

Example The following line displays the version of REALbasic that is running.

```
MsgBox "You're running version "+Str(RBVersion)+" of REALbasic!"
```

Note When you build your application, you can enter version information about your application in the Build Settings dialog box. This information is stored in your application's 'vers' resource. For more information, see the chapter on building applications in the *User's Guide*.

See Also TargetMacOS, Target68K, TargetPPC, TargetCarbon, TargetWin32, DebugBuild, RBVersionString constants.

RBVersionString Constant

Returns the version of REALbasic as a string.

Syntax

***result*=RBVersionString**

Part	Type	Description
result	String	Version of REALbasic that is running.

Note

When you build your application, you can enter version information about your application in the Build Settings dialog box. This information is stored in your application's 'vers' resource.

Example

The following line displays the version of REALbasic that is running:


```
MsgBox "You're running version "+Rbversionstring+" of REALbasic!"
```

See Also

TargetMacOS, Target68K, TargetPPC, TargetCarbon, TargetWin32, DebugBuild, RBVersion constants.

REALDatabase Class

A Database object that represents a REALDatabase back end. Use the **REALDatabase** class to open or create REALdatabases programmatically.

Super Class

Database class.

Properties

Name	Type	Description
DatabaseFile	FolderItem	The FolderItem for the REALDatabase file.

Methods

Name	Parameters	Description
CreateDataBaseFile		Creates a new REALdatabase. It uses the DatabaseFile property as the FolderItem for the database to create. Returns a Boolean—True if the new database was created successfully, and False if otherwise.

Notes

A REAL database is single-user and is limited to 8,174 bytes per record, 8,174 bytes per varchar field, and 254 columns per table. There is no fixed maximum number of rows.

REALDatabase should be used instead of OpenREALDatabase. OpenREALDatabase may no longer be supported in future releases of REALbasic.

Examples

The following example opens an existing REALdatabase.

```
Dim dbFile as FolderItem
Dim db as REALDatabase
db=New REALDatabase
dbFile = GetFolderItem("Pubs")
db.DatabaseFile=dbFile
If db.Connect() then
 //proceed with database operations here..
else
 Beep
 MsgBox "The database couldn't be opened."
end if
```

The following example creates a new REALdatabase:

```
Dim db as REALDatabase
Dim f as FolderItem
Dim result as Boolean
f=New FolderItem("mydb")
db=New REALDatabase
db.databaseFile=f
result=db.CreateDatabaseFile
If result then
  //proceed to do great things here...
else
  MsgBox "Database not created"
end if
```

See Also

NewREALdatabase, Open4DDatabasebyADSP, Open4DDatabasebyTCPIP, OpenCSVCursor, OpenDBFCursor, OpenDTFDatabase, OpenODBCDatabase, OpenOracleDatabase, OpenOpenBaseDatabase, OpenPostgreSQLDatabase,

OpenREALDatabase, Select4DDatabaseByADSP, Select4DDatabaseByTCPIP, SelectODBCDatabase functions; Database, RecordSet classes.

RecordSet Class

A **RecordSet** is a group of Database records. Use the methods and properties of the **RecordSet** class to navigate among this set of records and edit and update individual records. **RecordSet** offers the same functionality as did DatabaseCursor in previous releases of REALbasic. However, functions that returned a DatabaseCursor in previous releases now return a **RecordSet**.

DatabaseCursor is maintained in REALbasic for backwards compatibility. In future releases of REALbasic, DatabaseCursor may no longer be supported.

Super Class

Object

Properties

Name	Type	Description
BOF	Boolean	Beginning of the set of records.
EOF	Boolean	End of the set of records
FieldCount	Integer	Number of fields in the **RecordSet**
RecordCount	Integer	The number of records in the **RecordSet**. Currently supported in OpenBase, MySQL, PostgreSQL, and 4D plugins. For databases that do not support function, RecordCount returns -1.

Methods

Name	Parameters	Description
Close		Closes an open **RecordSet**. If you quit the application without calling Close, REALbasic does an implicit Close.
MoveFirst		Moves the record pointer to the first record in the **RecordSet**. Supported only for the REALdatabase data source.
MoveLast		Moves the record pointer to the last record in the **RecordSet**. Supported only for the REALdatabase data source.
MoveNext		Moves the record pointer to the next record in the **RecordSet**. Supported for all data sources.
MovePrevious		Moves the record pointer to the previous record in the **RecordSet**. Supported only for the REALdatabase data source.
Field	Name as String	Returns value of the Field in the row the record pointer is pointing to.

Name	Parameters	Description
IdxField	Index as Integer	1-based array. Use to refer to i^{th} field in the **RecordSet**.
Edit		Call prior to performing modifications to the current record.
Update		Call to update **RecordSet** to reflect changes to the record the record pointer is pointing to.
DeleteRecord		Deletes the record in the **RecordSet** the record pointer is pointing to.

Notes

If you are using REALbasic as a front-end in a multi-user environment, keep in mind the following. The Edit method will attempt to lock the current record to other users. If a user tries to access a locked record, the Database class Error property will return True.

The MoveNext method unlocks the current record if you previously locked it with the Edit method. It also calls the Close method of the class and the Close method of the Database class.

If you are using OpenBase as your data source and use the **RecordSet** class to update a record using Edit and Update you need to include the "_rowid" column of the table. If you don't, OpenBase won't be able to find the record in order to update it.

Examples

The following example modifies a field and updates the table.

```
Dim db as Database
dim rs as RecordSet
.
.
rs = db.SQLSelect("select * from employees where ID=01")
rs.Edit
rs.IdxField(2).setString ("VP")
rs.Update
db.Commit
```

See Also

Database, DatabaseField, ODBCDatabase ADSP4DServer, TCPIP4DServer, PostgreSQLDatabase, OpenBaseDatabase classes.

Rectangle Control 

Draws a rectangle.

Super Class RectControl

Because this is a RectControl, see the RectControl for other properties and events that are common to all RectControl objects.

Properties

Name	Type	Description
BorderWidth	Integer	The width of the rectangle's border in pixels.
BottomRightColor	Color	The color of the bottom and right edges of the rectangle.
FillColor	Color	The color of the inside of the rectangle.
LeftTopColor	Color	The color of the left and top edges of the rectangle.

Events

Name	Parameters	Description
MouseDown	x as Integer, y as Integer	The mouse button was pressed inside the Rectangle region at the location passed in to x,y. Return True if you are going to handle the MouseDown.
MouseDrag	x as Integer, y as Integer	The mouse button was pressed inside the Rectangle and moved (dragged) at the location local to the control passed in to x,y. This event will not occur unless you return True in the MouseDown event.
MouseUp	x as Integer, y as Integer	The mouse button was released inside the Rectangle region at the location passed in to x,y. This event will not occur unless you return True in the MouseDown event.

Example

You can give a rectangle a sunken or raised appearance using the global functions LightBevelColor, DarkBevelColor, and FillColor.

```
//Sunken Appearance
Me.BorderWidth=2
Me.TopLeftColor=DarkBevelColor
Me.BottomRightColor=LightBevelColor
Me.FillColor=FillColor

//Raised Appearance
Me.BorderWidth=2
Me.TopLeftColor=LightBevelColor
Me.BottomRightColor=DarkBevelColor
Me.FillColor=FillColor
```

See Also

RoundRectangle control; RectControl class.

RectControl Class

RectControl is the base class for most other control classes except the Line control. This means that all controls inherit the properties of the **RectControl**. **RectControls** cannot be created or modified directly. Instead, you create and modify the **RectControl** subclasses like PushButtons, EditFields, ListBoxes, etc.

Super Class

Control

Because this is a Control, see the Control Class for other properties and events that are common to all Control objects.

Properties

Name	Type	Description
Active	Boolean	Indicates whether the RectControl is active. Active is False when the RectControl's window is not in the foreground. When a window is deactivated, its controls are automatically deactivated unless AutoDeactivate is set to False.
AutoDeactivate	Boolean	Determines whether the control should be deactivated (on Mac OS) when the parent window is deactivated. Default is True.
Balloon Help	String	Balloon Help text for the control in Mac OS "classic" and Windows or the extended help tag in Mac OS X.

Name	Type	Description
Parent	Control	Used to get and set the control's parent control or window. Returns Nil if the parent is the window. Assign Nil to make the control's parent the window, even if it is enclosed by another control. The child control's behaviour in the IDE (see "Control Hierarchy" in Notes, below) will reflect the parent's state. If the parent control is somehow in another window, an InvalidParentException will occur.
DisabledBalloonHelp	String	The text that appears when the user moves the mouse over the control while the control is disabled and BalloonHelp is on.
Enabled	Boolean	Determines if the control should be enabled when the owning window is opened. A disabled control cannot be clicked and cannot receive the focus.
HelpTag	String	Text of help message displayed as a Windows "tip" or Carbon/Mac OS X help tag. It works in the Carbon version of the REALbasic IDE and in all Carbon applications. The tip/tag is displayed when the user places the mouse on the control and leaves it there.
Height	Integer	The height (in pixels) of the control.
Left	Integer	The left side of the control in local coordinates (relative to the window).
LockBottom	Boolean	Determines whether the bottom edge of the control should stay at a set distance from the bottom edge of the owning window.
LockLeft	Boolean	Determines whether the left edge of the control should stay at a set distance from the left edge of the owning window. *LockLeft* has no effect unless *LockRight* is True.
LockRight	Boolean	Determines whether the right edge of the control should stay at a set distance from the right edge of the owning window.
LockTop	Boolean	Determines whether the top edge of the control should stay at a set distance from the top edge of the owning window. *LockTop* has no effect unless *LockBottom* is True.
Top	Integer	The top of the control in local coordinates (relative to the window).
MouseCursor	MouseCursor	The cursor to be displayed while the mouse is within the control and both the Application and Window classes's MouseCursor properties are Nil. If the Application class's MouseCursor property is not Nil or the Window's MouseCursor property is not Nil, then any control's MouseCursor property is ignored. Names are: WatchCursor, IBeamCursor, and ArrowCursor. You can also obtain a MouseCursor from a resource file.

Name	Type	Description
Visible	Boolean	Determines if the control should be visible when the owning window is opened.
Width	Integer	The width (in pixels) of the control.
Window	Window	The RectControl's parent window.

Events

Name	Parameters	Description
DropObject	Obj as DragItem	The item represented by *Obj* has been dropped on the control.
MouseEnter		The mouse has entered the area of the control.
MouseExit		The mouse has left the area of the control.
MouseMove	X as Integer, Y as Integer	The mouse has moved within the control to the coordinates passed. The coordinates are local to the control, not to the window.

Methods

Name	Parameters	Description
AcceptFileDrop	FileType as String	Permits documents of type *FileType* to be dropped on the control. *FileType* must be a file type that you defined in the File Types dialog box.
AcceptMacDataDrop	MacType as String	Permits data (of the Type specified) to be dropped on the control.
AcceptPictureDrop		Permits pictures to be dropped on the control.
AcceptTextDrop		Permits text to be dropped on the control.
NewDragItem	Left As Integer, Top As Integer, Width As Integer, Height As Integer	Defines the drag rectangle (based on the coordinates passed) that will appear when the user drags from the control. Returns a new DragItem object. *Left* and *Top* are the coordinates of the left-top corner of the drag rectangle and *Width* and *Height* are the dimensions of the drag rectangle. See the NewDragItem global function, which provides equivalent functionality, as well as support for dragging to a window.
Refresh		Repaints the entire contents of the control.
RefreshRect	X As Integer, Y As Integer, Width As Integer, Height As Integer	Repaints only the region of the control specified.

Name	Parameters	Description
SetFocus		If applicable, sets the focus to the control. Keydown events are directed to the control. For EditFields, the insertion point is moved to the control. If the control cannot get the focus on the platform on which the application is running, SetFocus does nothing. On Macintosh, only EditFields, Canvas controls, and ListBoxes get the focus. On Windows, EditFields, PushButtons, PopupMenus, CheckBoxes, ListBoxes, and Sliders can get the focus. The SetFocus method of the Window class or the ClearFocus method can be used to remove the focus from the control that currently has the focus, leaving no control with the focus.

Notes

The Control Hierarchy

In some cases you build portions of your interface by placing some controls within another control. For example, you use the GroupBox control to organize other controls, usually RadioButtons or possibly Checkboxes. The TabPanel and PagePanel controls also designed to enclose other controls. A good example is the Find and Replace dialog box in the REALbasic *Tutorial*. In this case, StaticText, EditField, and PushButton controls are all enclosed by the TabPanel.

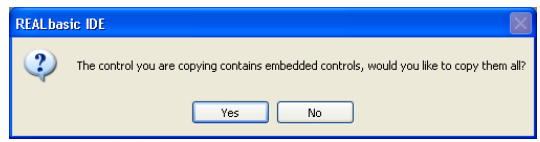
The control that encloses the others is known as the *parent* control and the controls that are entirely within its borders are the *child* controls. This works automatically if you create the controls in the order of: parent-child. That is, create the control that encloses the others first, then create the child controls or duplicate existing child controls and keep them inside the parent control.

You can use the Parent property of the RectControl class to either get or set the parent of an existing control. If you create a child control before its parent, you can set its parent using its Parent property.

If a control is not completely enclosed by another control then it doesn't automatically become a child; it simply overlaps the other control.

If you move a child outside its parent, it is no longer a child of that control. If you move it completely inside another control, it becomes the child of that control.

When you copy a parent control, you can copy all its child controls as well. When you do the copy, REALbasic displays an alert box in which you can elect to copy all the child controls.



You can create more than one level of nesting. For example, you can place a GroupBox within a TabPanel and then place several RadioButtons within the GroupBox. In this case, the GroupBox is the parent of the RadioButtons and the TabPanel is the parent of the GroupBox. To make this work automatically, you must create them in the order that respects the hierarchy: First create the TabPanel, then the GroupBox, and then the RadioButtons.

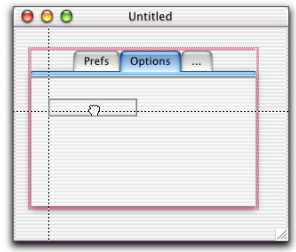
Note Obviously, the parent control must be in the same window as its child controls. If you attempt to get the parent of a control whose parent is in another window, you will get an InvalidParentException error.

If a control is not enclosed by another control, then its "parent" is its window. But, since the Parent property of a control is a **Control**, the Parent property will be Nil—since a Window is not subclassed from the **Control** class. Before trying to access the Parent property of a control, be sure to first test whether it is Nil. If you don't, you will get a NilObjectException error if the control's "parent" is the parent window.

If a control is enclosed by another control, but you don't want it to behave as a child, then set its Parent property to Nil. This breaks the default Parent-Child relationship that is normally maintained by the control hierarchy.

Control Hierarchy Features

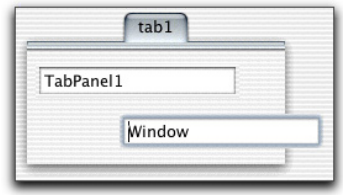
To take advantage of the features of the control hierarchy automatically, create the parent control first and then add child controls. When you add a child to a parent in this manner, a marquee surrounds the parent. This gives you visual confirmation that you are, in fact, adding a child to a parent. For example, in this illustration a StaticText is being added as a child to a TabPanel.



A marquee also surrounds the parent control whenever you select an existing child control. You can disable the marquee by disabling the "Highlight parent control" preference on the Window Editor panel in REALbasic ▶ Preferences. If you leave the feature enabled, you can also select the color of the marquee in that preference panel.

If you duplicate a child control and leave the duplicate within the parent, then it is automatically a child. However, if you move the duplicate outside the parent, it is no longer a child. A control must be fully enclosed by the parent to be considered a child.

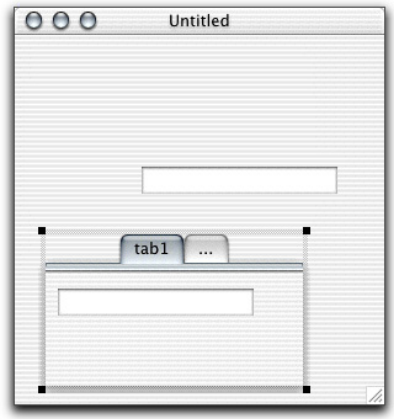
In the following example, both EditFields report the identity of their parents. The bottom one was duplicated from the top one, but has been moved out of the TabPanel. It is no longer a child.



When you move a child out of the 'scope' of the parent, the marquee surrounding the parent is turned off, indicating that the child is not under parental control anymore.

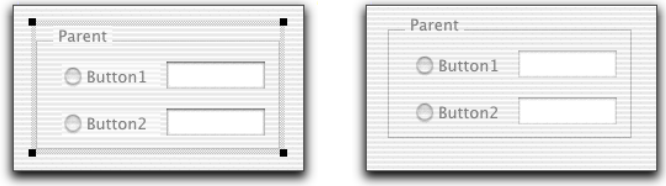
If you move the bottom EditField back within the TabPanel, it becomes a child of the TabPanel once again.

Moving a parent control moves its child controls as well. In the following example, the TabPanel shown above was moved down. It takes the top EditField with it, but leaves the bottom one orphaned.



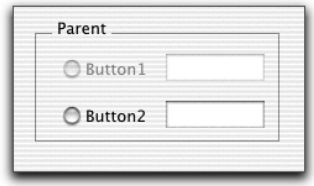
- Deleting a parent control deletes all child controls.
- Hiding a parent hides all child controls, but retains the previous visibility status of all children.
- Showing a parent control shows only the child controls whose visibility is set.
- Disabling a parent control disables all its child controls, but retains the previous visibility status of all children. In the IDE, disabling a container visually disables all the child controls, but does not update the Enabled property.

Here is a disabled GroupBox in the IDE in which two RadioButtons and two EditFields are children. When the GroupBox is disabled, all four children are disabled. Since all children are disabled when the parent is disabled, the EditFields are not enterable in the application.



Development environment Application

Enabling a parent control enables only the child controls whose Enabled property is set. In the following example, only the bottom row of child controls have the Enabled property set. When the GroupBox is enabled, the top row of controls remains disabled.



Disabling the GroupBox disables all child controls, as expected.

The Window Class's Composite Property

Under Mac OS X, there is a display problem that affects certain types of **RectControls** that are placed in Metal or Drawer windows or on top of other controls in Document windows. Non-rectangular shaped controls such as PushButtons or RadioButtons show their rectangular borders as a white background. To solve this problem, turn on the parent window's Composite property.

See Also

Control class.

RectShape Class

Draws a (two-dimensional) rectangle in a vector graphics environment.

SuperClass

Object2D

Properties

Name	Type	Description
Height	Double	The height of the rectangle.
Width	Double	The width of the rectangle.

Methods

Name	Parameters	Description
Contains	X as Double, Y as Double	Returns a Boolean. Returns True if the rectangle contains the point specified by x,y.

Example

The following code in the Paint event of a window draws a diamond shaped rectangle.

```
Dim r as RectShape
r=New RectShape
r.width=75
r.height=75
r.border=100
r.bordercolor=RGB(0,0,0) //black
r.fillcolor=RGB(0,127,127) // teal
r.borderwidth=2.5
r.rotation=-.78
g.DrawObject r,100,100
```

See Also

ArcShape, CurveShape, FigureShape, FolderItem, Group2D, Graphics, OvalShape, Picture, PixmapShape, RoundRectShape, StringShape classes.

ReDim Method

Resizes the array property or variable specified.

Syntax

ReDim *arrayName*(*newSize1*,*[newSizeN]*)

Part	Type	Description
arrayName	Variable name	The name of any previously declared array.
newSize1	Integer	The new size for the array.
newSizeN	Integer	The new size for the N^{th} dimension of the array.

Notes

The **Redim** method is used to increase or reduce the number of elements in the array specified. Arrays are zero-based (the first element is zero) so you resize the array using a number that is one less than the number of elements you actually want.

The maximum number of dimensions in a multi-dimensional array is 20.

Examples

This example reduces the aNames array to 11 elements.

```
Redim aNames(10)
```

This example adds 10 elements to the aNames array.

```
Redim aNames(UBound(aNames)+10)
```

This example reduces the aPeople array to 11 elements for the first dimension and 6 elements for the second dimension.

```
Redim aPeople(10,5)
```

See Also

Append method, Dim function, Insert method, Remove method, UBound function, Sort method.

RegistryAccessErrorException Error

Occurs if you try to use the RegisteryItem class without proper access privileges or try to use it under any Macintosh OS. Registry items is a Windows-only feature.

Super Class

RuntimeException

See Also

RegisteryItem class, Exception Block.

RegistryItem Class

Creates and manages registry items on Windows. Does not apply to Macintosh.

Super Class

Object

Constructors

Name	Parameters	Description
RegisteryItem	Path as String	Creates the registry item from the path passed to it. Returns a new **RegisteryItem**.
RegisteryItem	Copy as **RegisteryItem**	Clones the **RegisteryItem** passed to it. Returns a new **RegisteryItem**.

Properties

Name	Type	Description
FolderCount	Integer	The number of child folders contained with the **RegisteryItem**.
KeyCount	Integer	The number of keys contained within the **RegisteryItem**.
Parent	**RegisteryItem**	The RegisteryItem's parent. Note: You cannot get the parent of a hive. Each hive is considered a root.
Path	String	The full path to the **RegisteryItem**.

Methods

Name	Parameters	Description
Child	Name as String	Returns the child named as a **RegisteryItem**. Returns a **RegisteryItem**.
DefaultValue		Returns the default value of the **RegisteryItem** as a Variant.
Item	Index as Integer	Returns as a **RegisteryItem** the **RegisteryItem** corresponding to *Index*. If *Index* is greater than KeyCount, then an OutofBoundsException is raised.
Name	Index as Integer	Returns as a String the name of the RegisteryItem corresponding to *Index*. If *Index* is greater than KeyCount, then an OutofBoundsException is raised.
Value	Name as String	Returns the **RegisteryItem**'s value as a Variant.
Value	Index as Integer	Returns the **RegisteryItem**'s value as a Variant. Index refers to a Key contained in the RegisteryItem. If *Index* is greater than KeyCount, then an OutofBoundsException is raised.

See Also

RegisteryAccessErrorException error.

RegEx Class

Used to do search and replace operations using regular expressions (i.e., perl).

Super Class

Object

Properties

Name	Type	Description
Options	RegExOptions	These options are various states which you can set for the Regular Expressions engine. See the RegExOptions class.
ReplacementPattern	String	This is the replacement string, which can include references to substrings matched previously, via the standard '\1' or '$1' notation common in regular expressions. This pattern is used either with the Replace property or passed to the RegExMatch class when Search returns, and subsequently used with Replace if no parameters are specified.
SearchPattern	String	This is the pattern you are currently searching for.
SearchStartPosition	Integer	Character position at which you want to start the search if the optional TargetString parameter to *Replace* is not specified. Keep in mind if you set it, it will only be used if you don't specify a TargetString, since setting a new targetString resets the value.

Methods

Name	Parameters	Description
Replace	Optional: TargetString as String, SearchStartPosition as Integer	Finds *SearchPattern* in Target, Replaces the contents of *SearchPattern* with *ReplacementPattern*. Returns the resulting string. Replace can take the optional parameters shown at left. Returns a String.
Search	TargetString as String, SearchStartPosition as Integer	Finds *SearchPattern* in *TargetString*, if it succeeds it returns a RegExMatch. The RegExMatch will remember the ReplacementPattern specified at the time of the search. Returns a RegExMatch.

Notes

This section describes the syntax of regular expressions in REALbasic.

Pattern	Description
.	Matches any character except newline.
[a-z0-9]	Matches any single character of set.
[^a-z0-9]	Matches any single character not in set.

Pattern	Description
\d	Matches a digit. Same as [0-9].
\D	Matches a non-digit. Same as [^0-9].
\w	Matches an alphanumeric (word) character — [a-zA-Z0-9_].
\W	Matches a non-word character [^a-zA-Z0-9_].
\s	Matches a whitespace character (space, tab, newline, etc.).
\S	Matches a non-whitespace character.
\n	Matches a newline (line feed).
\r	Matches a return.
\t	Matches a tab.
\f	Matches a formfeed.
\b	Matches a backspace.
\0	Matches a null character.
\000	Also matches a null character because of the following:
\n*nn*	Matches an ASCII character of that octal value.
\x*nn*	Matches an ASCII character of that hexadecimal value.
\c*X*	Matches an ASCII control character.
metachar	Matches the meta-character (e.g., \, ., l).
(abc)	Used to create subexpressions. Remembers the match for later backreferences. Referenced by replacement patterns that use \1, \2, etc.
\1, \2,…	Matches whatever first (second, and so on) of parens matched.
x?	Matches 0 or 1 *x*'s, where x is any of above.
*x**	Matches 0 or more *x*'s.
x+	Matches 1 or more *x*'s.
x{*m*,*n*}	Matches at least *m* *x*'s, but no more than *n*.
abc	Matches all of a, b, and c in order.
alblc	Matches one of a, b, or c.
\b	Matches a word boundary (outside [] only).
\B	Matches a non-word boundary.
^	Anchors match to the beginning of a line or string.
$	Anchors match to the end of a line or string.

Replacement Patterns

The following expressions can only apply to the replacement pattern:

Pattern	Description
$`	Replaced with the entire target string before match.
$&	The entire matched area; this is identical to \0 and $0.
$'	Replaced with the entire target string following the matched text.

Pattern	Description
$0-$50	$0-$50 evaluate to nothing if the the subexpression corresponding to the number doesn't exist.
\0-\50	
\x*nn*	Replaced with the character represented by *nn* in Hex, e.g., \xAA is ™.
\n*nn*	Replaced with the character represented by *nn* in Octal.
\c*X*	Replaced with the character that is the control version of *X*, e.g., \cP is DLE, data line escape.

Double-byte Systems

If you are working with a double-byte system such as Japanese, RegEx cannot operate on the characters directly. You should first convert all double-byte text to UTF8 using REALbasic's built-in Text Converter functions. See, for example, the TextConverter class for an example of how to use the Text Converter functions.

All text that will be processed by RegEx should be converted. This includes *SearchPattern*, *ReplacementPattern*, and *TargetString*. The result of the Search or Search and Replace will be a UTF8 string, so you will need to convert it back to its original form using the Text Converter functions. Both Search and Search and Replace operations work on both Macintosh and Windows, provided that this conversion takes place.

Regular Expression Examples

The basic idea of regular expressions is that it enables you to find and replace text that matches the set of conditions you specify. It extends normal Search and Replace with pattern searching.

Wildcards

Some special characters are used to match a class of characters:

Wildcard	Matches
.	Any single character except a line break, including a space.

If you use the "." as the search pattern, you will select the first character in the target string and, if you repeat the search, you will find each successive character, except for Return characters

The following wildcards match by position in a line:

Wildcard	Matches	Example
^	Beginning of a line (unless used in a character class; see below)	^Phone: Finds lines that begin with "Phone":
$	End of a line (unless used in a character class)	$: Finds the last character in the current line.

Character Classes

A character class allows you to specify a set or range of characters. You can choose to either match or ignore the character class. The set of characters is enclosed in brackets.

If you want to ignore the character class instead of match it, precede it by a caret (^). Here are some examples:

Character Class	Matches
[aeiou]	Any one of the characters a, e, i, o, u.
[^aeiou]	Any character except a, e, i, o, u.
[a-e]	Any character in the range a-e, inclusive
[a-zA-Z0-9]	Any alphanumeric character. Note: Case-sensitivity is controlled by the CaseSensitive property of the RegExOptions class.
[[]	Finds a [.
[]]	Finds a]. To find a a closing bracket, place it immediately after the opening bracket.
[a-e^]	Finds a character in the range a-e or the caret character. To find the caret character, place it anywhere except as the first character after the opening bracket.
[a-c-]	Finds a character in the range a-c or the - sign. To match a -, place it at the beginning or end of the set.

Non-printing Characters

You can use the following notation to find non-printing characters:

Special Character	Matches
\r	Line break (return)
\n	Newline (line feed)
\t	Tab
\f	Formfeed (page break)
\x*NN*	Hex code *NN*.

Other Special Characters

The following patterns are wildcards for the following special characters:

Special Character	Matches
\s	Any whitespace character (space, tab, return, linefeed, form feed)
\S	Any non-whitespace character.
\w	Any "word" character (a-z, A-Z, 0-9, and _)
\W	Any "non-word" character (All characters not included by \w).
\d	Any digit [0-9].
\D	Any non-digit character.

Repetition Characters

Repetition characters are modifiers that allow you to repeat a specified pattern.

Repetition Character	Matches	Examples
*	Zero or more characters.	d* finds no characters, or one or more consecutive "d"s. .* finds an entire line of text, up to but not including the return character.
+	One or more characters.	d+ finds one or more consecutive "d"s. [0-9]+ finds a string of one or more consecutive numbers, such as "90404", "1938", the "32" in "Win32", etc.
?	Zero or one characters.	d? finds no characters or one "d".

Please note that, since * and ? match zero instances of the pattern, they always succeed but may not select any text. You can use them to specify an optional character, as in the examples in the following section.

"Greediness"

REALbasic supports the "?" as a "greediness" modifier for a subpattern in a regular expression. By default, greediness is controlled by the Greedy property of the RegExOptions class, but can be overridden using the "?". You can place a "?" directly after a * or + to reverse the "greediness" setting. That is, if Greedy is True, using the ? after a * or + causes it to match the minimum number of times possible: For example, consider the following.

Target String	Greedy	Regular Expression	Result
aaaa	True	(a+?) (a+)	$1=a, $2=aaa
aaaa	False	(a+?) (a+)	$1=aaa, $2=a

Extension Mechanism

We also support the regular expression extension mechanism used in Perl. For instance:

(?#text)	Comment
(?:pattern)	For grouping without creating backreferences
(?=pattern)	A zero-width positive look-ahead assertion. For example, \w+(?=\t) matches a word followed by a tab, without including the tab in $&.
(?!pattern)	A zero-width negative look-ahead assertion. For example foo(?!bar)/matches any occurrence of "foo" that isn't followed by "bar".
(?<=pattern)	A zero-width positive look-behind assertion. For example, (?<=\t)\w+ matches a word that follows a tab, without including the tab in $&. Works only for fixed-width look-behind.
(?<!pattern)	A zero-width negative look-behind assertion. For example (?<!bar)foo matches any occurrence of "foo" that does not follow "bar". Works only for fixed-width look-behind.

Subexpressions

You can use parentheses within your search patterns to isolate portions of the matched string. You do this when you need to refer to subsections of the matched in your

replacement string. For example you would do this if you need to replace only a portion of the matched string or insert other text into the matched string.

Here is an example. If you want to match any date followed by the letters "B.C." you can use the pattern "\d+\sB\.C\." (Any number of digits followed by a space character, followed by the letters "B.C.") This will match dates such as 33 B.C., 1742 B.C., etc. However, if you wanted your replacement pattern to leave the year alone but replace the letters with something else, you would use parens. The search pattern "(\d+)\s(B\.C\.)" does this.

When you write your replacement pattern, you can refer to the year only with the variable \1 and the letters with \2.

If you write "(\d+)\s(B.C.|A.D.|BC|AD)", then \2 would contain the matched letters.

Combining Patterns

Much of the power of regular expressions comes from combining these elementary patterns to make up complex searches. Here are some examples:

Pattern	Matches
\$?[0-9,]+\.?\d*	Matches dollar amounts with an optional dollar sign.
\d+\sB\.C\.	one or more digits followed by a space, followed by "B.C."

The Alternation Operator

The alternation operator (|) allows you to match any of a number of patterns using the logical "or" operator. Place it between two existing patterns to match either pattern. You can use more than one alternation operator in a pattern:

Pattern	Matches
\she\s | \sshe\s	" he " or " she "
cat|dog|possum	"cat", "dog", or "possum"
([0-9,]+\sB\.C\.)|([0-9,]+\sA\.D\.) *or* [0-9,]+\s((B\.C\.)|(A\.D\.))	Years of the form "*yearNum* B.C. *or* A.D." e.g., "2,175 B.C." or "215 A.D."

Search and Replace

You use special patterns to represent the matched pattern. Using replacement patterns, you can append or prepend the matched pattern with other text.

Pattern	Description
$&	Contains the entire matched pattern. If "\d\d\d\d\sB\.C\." finds "1541 B.C.", then the replacement pattern "the year $&" results in "the year 1541 B.C.", as the $& contains the string "1541 B.C".
\1, \2, etc.	Contains the matched subpatterns, defined by use of parentheses in the search string. The search pattern "(\d+)\s(B\.C\.|A\.D\.|BC|AD)" looks for any number of digits followed by a space character, followed by either "B.C.", "BC", "A.D.", or "AD". The \1 variable contains the match to the "\d+" portion of the expression and \2 contains the match to the "B\.C\.|A\.D\.|BC|AD" portion.

Credits

REALbasic uses a modified version of the PCRE library package, which is open source software, written by Philip Hazel, and copyright by the University of Cambridge, England.

The source to this library is available at:
ftp://ftp.csx.cam.ac.uk/pub/software/programming/pcre/

Examples

The following PushButton's Action event handler allows you to search the text in EditField1 using the search pattern entered into EditField2 and display the results of the search in a StaticText:

```
Dim rg as RegEx
Dim myMatch as RegExMatch
rg=new RegEx
rg.SearchPattern=EditField2.text
myMatch=rg.search(EditField1.text)
if myMatch <> Nil then
 StaticText1.text=myMatch.SubExpressionString(0)
else
 StaticText1.text="Text not found!"
End if
exception err as RegExException
 MsgBox err.message
```

See Also

RegExMatch, RegExOptions classes; RegExException Error.

RegExException Error

The RegEx engine found a runtime error in your code. Currently, the only type of runtime exception is a RegExSearchPatternException, but future versions of REALbasic may trap other types of errors.

Super Class RuntimeException

See Also

RuntimeException class; RegExSearchPatternException error.

RegExMatch Class

Used to extract the matched string when doing a search with regular expressions.

Super Class

Object

Properties

Name	Parameters	Description
SubExpressionCount		Number of SubExpressions that are available with the search just performed. REALbasic's Regular Expressions support both \number and $number syntax for SubExpressions. SubExpressions allow replacement of parts of the pattern. Returns an Integer.
SubExpressionString	matchNumber as Integer	Returns the SubExpression as a string for the given matchNumber. 0 returns the entire MatchString (the implicit 0^{th} subExpression), and 1 is the first real subExpression. Returns a String.
SubExpressionStartB	matchNumber as Integer	Returns the starting byte position of the subExpression given by *matchNumber*. Returns an Integer.
Replace	ReplacementPattern as String (Optional)	Substitutes the matched result in a manner specified by the given *ReplacementPattern*. If no *ReplacementPattern* is specified, it uses the ReplacementPattern which was specified in the RegEx object at the time of the search. Returns a String.

See Also

RegEx class.

RegExOptions Class

Used to specify options when doing a search using regular expressions.

Super Class

Object

Properties

Name	Type	Description
CaseSensitive	Boolean	Specifies if case is to be considered when matching a string. Default is False.

Name	Type	Description
DotMatchAll	Boolean	Normally the period matches everything except a new line, this option allows it to match new lines. Default is False.
Greedy	Boolean	Greedy means the search finds everything from the beginning of the first delimiter to end of the last delimiter and everything in-between. For example, Say you want to match the following bold-tagged text in HTML: The quick brown fox jumped If you use this pattern: .+ You end up matching "quick brown fox", which isn't what you wanted. So, you can turn *Greedy* off or use this syntax: .+? and you will match "quick", which is exactly what you wanted.) Default is True.
LIneEndType	Integer	This is in effect for the current Regular Expression "session". Has no effect on SearchPatterns if *TreatTargetAsOneLine* is True. Changes the way \n is expanded for ReplacementPatterns 0 = any line ending (Mac, Win32, or Unix) 1 = default for platform (if running on Mac same as 2) (if running on Win32 same as 3) 2 = Mac ASCII 13 or \r 3 = Win32 ASCII 10 or \n 4 = Unix ASCII 10 or \n Default is 0.
MatchEmpty	Boolean	Indicates whether patterns are allowed to match the empty string. Default is True.
ReplaceAllMatches	Boolean	Indicates whether all occurrences of the pattern are to be replaced. Default is False.
StringBeginIsLineBegin	Boolean	Indicates whether a string's beginning should be counted as the beginning of a line. Default is True.
StringEndIsLineEnd	Boolean	Indicates whether a string's end should be counted as the end of a line. Default is True.
TreatTargetAsOneLIne	Boolean	Ignores internal newlines for purposes of matching against '^' and '$'. Default is False.

See Also

RegEx, RegExMatch classes

RegExSearchPatternException Error

You used an invalid regular expression search pattern.

Super Class RegExException error.

Example The following exception block, placed at the end of a method that does a regular exception search, traps for invalid search patterns and prints the Message property of the RegExSearchPatternException. You could also use RegExException, since RegExSearchPatternException is subclassed from RegExException.

```
Exception err as RegExSearchPatternException
MsgBox err.message
```

See Also RuntimeException class.

Rem Statement

Used to add comments to your code.

Syntax **Rem** ***text string***

Notes REALbasic's compiler will ignore any comments you enter using the **Rem** statement. Comments are not included in stand-alone applications.

The ' (apostrophe) or two forward slashes (//) can be used to comment code as well. Comments can appear on separate lines or on the same line as executable code, provided they are to the right of any code that will execute. Valid comments appear in the Code Editor in red.

The Command-' keyboard shortcut toggles the selected lines of code between commented out and 'live' states.

Example The following code uses comments to document the results of Boolean comparisons:

```
Dim a, b, c, d As Boolean
a=True
b=False
d=a Or b Rem Evaluates to True
d=b And c Rem Evaluates to False
```

See Also //, ' operators.

Remove Method

Removes an element from an array.

Syntax

array*.Remove *index

Part	Type	Description
array	Any data type	Required. The array to remove the element from.
index	Integer	Required. The element number to be removed.

Notes

The **Remove** method removes the *index* element from the *array*.

The **Remove** method works with one-dimensional arrays only.

Example

This example removes element 4 (The Fifth Element) from the aNames array:

```
aNames.Remove
```

See Also

Append method, Dim statement, Insert method, Redim method, UBound function, Sort method.

Replace Function

Replaces the first occurrence of a string with another string.

Syntax

***result*=Replace(*sourceString*, *oldString*, *newString*)**

OR

***result*=*stringVariable*.Replace(*oldString*, *newString*)**

Part	Type	Description
result	String	A copy of *sourceString* with any changes made by the Replace function.
sourceString	String	The original string.
oldString	String	The characters to be replaced.
newString	String	The replacement characters.
stringVariable	String	Any variable of type String.

Notes

Replaces the first occurrence of *oldString* in *sourceString* with *newString*. **Replace** is case-insensitive.

If *newString* is an empty string (""), the **Replace** function deletes the first occurrence of the *oldString* in the *sourceString*.

If *oldString* is an empty string (""), the **Replace** function returns an unchanged copy of the *sourceString*.

Examples

Below are some examples that show the results of the **Replace** function:

```
Dim result As String
result=Replace("The quick fox","fox","rabbit") //returns "The quick rabbit"
result=Replace("The quick fox","f","b") //returns "The quick box"
result=Replace("The quick fox","quick","") //returns "The fox"
```

Using the second syntax:

```
Dim result,s As String
s="The quick fox"
result=s.Replace("fox","rabbit") //returns "The quick rabbit"
MsgBox result
```

See Also

ReplaceAll, ReplaceAllB, ReplaceB functions.

ReplaceB Function

Replaces the first occurrence of a string with another string.

Syntax

***result*=ReplaceB(*sourceString*, *oldString*, *newString*)**

OR

***result=stringVariable*.ReplaceB(*oldString*, *newString*)**

Part	Type	Description
result	String	A copy of *sourceString* with any changes made by the Replace function.
sourceString	String	The original string.
oldString	String	The characters to be replaced.
newString	String	The replacement characters.
stringVariable	String	Any variable of type String.

Notes

Replaces the first occurrence of *oldString* in *sourceString* with *newString*. **ReplaceB** is the binary version of Replace.

If *newString* is an empty string (""), the **ReplaceB** function deletes the first occurrence of the *oldString* in the *sourceString*.

If *oldString* is an empty string (""), the **ReplaceB** function returns an unchanged copy of the *sourceString*.

ReplaceB is case-sensitive; it treats *sourceString* as a series of raw bytes. It should be used instead of Replace when the string represents binary data or when your application will run in a one-byte character set (such as the US system) and you want case-sensitivity.

Examples

Below are some examples that show the results of the **ReplaceB** function:

```
Dim result As String
result=ReplaceB("The quick fox","fox","rabbit") //returns "The quick rabbit"
result=ReplaceB("The quick fox","f","b") //returns "The quick box"
result=ReplaceB("The quick fox","quick","") //returns "The fox"
```

Using the second syntax:

```
Dim result,s As String
s="The quick fox"
result=s.ReplaceB("fox","rabbit") //returns "The quick rabbit"
MsgBox result
```

See Also

ReplaceAll, ReplaceAllB, Replace functions.

ReplaceAll Function

Replaces all occurrences of a string with another string.

Syntax

***result*=ReplaceAll(*sourceString*, *oldString*, *newString*)**

OR

***result=stringVariable*.ReplaceAll(*oldString*, *newString*)**

Part	Type	Description
result	String	A copy of *sourceString* with any changes made by the ReplaceAll function.
sourceString	String	The original string.
oldString	String	The characters to be replaced.
newString	String	The replacement characters.
stringVariable	String	Any variable of type String.

Notes

The **ReplaceAll** function replaces all occurrences of *oldString* in *sourceString* with *newString*. **ReplaceAll** is case-insensitive.

If *newString* is an empty string (""), the **ReplaceAll** function deletes every occurrence of the *oldString* in the *sourceString*.

If *oldString* is an empty string (""), the **ReplaceAll** function returns an unchanged copy of the *sourceString*.

Examples

Below are some examples that show the results of the **ReplaceAll** function:

```
Dim result As String
result=ReplaceAll("xyxyxy","x","z") //returns "zyzyzy"
result=ReplaceAll("The quick fox"," ","") //returns "Thequickfox"
```

See Also

Replace, ReplaceAll, ReplaceAllB functions.

ReplaceAllB Function

Replaces all occurrences of a string with another string.

Syntax

***result*=ReplaceAllB(*sourceString*, *oldString*, *newString*)**

OR

***result=stringVariable*.ReplaceAllB(*oldString*, *newString*)**

Part	Type	Description
result	String	A copy of *sourceString* with any changes made by the ReplaceAll function.
sourceString	String	The original string.
oldString	String	The characters to be replaced.
newString	String	The replacement characters.
stringVariable	String	Any variable of type String.

Notes

The **ReplaceAllB** function replaces all occurrences of *oldString* in *sourceString* with *newString*. **ReplaceAllB** is case-sensitive because it treats the source string as a series of raw bytes.

If *newString* is an empty string (""), the **ReplaceAllB** function deletes every occurrence of the *oldString* in the *sourceString*.

If *oldString* is an empty string (""), the **ReplaceAllB** function returns an unchanged copy of the *sourceString*.

ReplaceAllB is case-sensitive; it treats *sourceString* as a series of raw bytes. It should be used instead of ReplaceAll when the string represents binary data or when your application will run in a one-byte character set (such as the US system) and you want case-sensitivity.

Examples

Below are some examples that show the results of the **ReplaceAll** function:

```
Dim result As String
result=ReplaceAllB("xyxyxy","x","z") //returns "zyzyzy"
result=ReplaceAllB("The quick fox"," ","") //returns "Thequickfox"
```

See Also

Replace, ReplaceB, ReplaceAll functions.

ReplaceLineEndings Function

Replaces the line endings in the passed String with the specified line ending.

Syntax

***result*=ReplaceLineEndings(*SourceString*,*LineEnding*)**

Part	Type	Description
result	String	A copy of *sourceString* with the new line endings replacing the original line endings.
SourceString	String	The original string.
LineEnding	String	The line ending used to replace the line endings in *SourceString*.

Notes

ReplaceLineEndings does a global search-and-replace for the line ending characters in *SourceString* using the specified line ending as the replacement string. The search automatically recognizes Windows, Macintosh, and Unix line endings. Use this function to make multiline (or multi-paragraph) text compatible across platforms. The easiest way to specify the replacement Line Ending is with the EndOfLine object.

Example

This example replaces the line endings in the text in an EditField with Macintosh line endings.

```
Dim s as String
s=ReplaceLineEndings(EditField1.Text,EndOfLine.Macintosh)
```

See Also

EndOfLine object.

ResourceFork Class

ResourceFork Class objects are used to read and write data to resources. You can import resource files into the Project Window (files of type "rsrc") by dragging them from the Finder.

Super Class

Object

Properties

Name	Parameters	Description
Geticl	ID as Integer	Loads large (32 x 32) icl icon specified by ID. Returns type Picture.
Getics	ID as Integer	Loads small (16 x 16) ics icon specified by ID. Returns type Picture.
ResourceLocked	Type as String, ID as Integer	Used to get and set the locked attribute of the resource.
ResourcePreload	Type as String, ID as Integer	Used to get and set the Preload attribute of the resource.
ResourceProtected	Type as String, ID as Integer	Used to get and set the Protected attribute of the resource.
ResourcePurgeable	Type as String, ID as Integer	Used to get and set the Purgeable attribute of the resource.
ResourceSysHeap	Type as String, ID as Integer	Used to get and set the SysHeap attribute of the resource.
TypeCount	Integer	Number of resources types present in the resource fork.

Methods

Name	Parameters	Description
AddPicture	Pict as Picture, ID as Integer, Name as String	Creates a PICT resource using the *Name* and ID specified and fills it with the *Pict* specified.
AddResource	Data as String, Type as String, ID as Integer, Name as String	Adds a resource of the Type specified, using the *Name* and *ID* specified and fills it with the *Data* specified.
Close		Closes the open resource fork. The resource fork will be closed automatically when the instance is destroyed.
GetCicn	ID as Integer	Returns the CICN (color icon) resource as a Picture based on the *ID* passed.

Name	Parameters	Description
GetCursor	ID as Integer	Returns the CURS resource corresponding to *ID* as an object of type MouseCursor.
GetHandle	Type as String ID as Integer	Returns a resource handle as an Integer, given the type and ID of the desired resource. Intended for use with Declare statements. The resource will be released when the resource fork is closed or when you call ReleaseHandle.
GetNamedPicture	Name as String	Returns the PICT resource as a Picture based on the *Name* passed.
GetNamedResource	Type as String, Name as String	Returns the specified resource as a string.
GetPicture	ID as Integer	Returns the PICT resource as a Picture based on the *ID* passed.
GetResource	Type as String, ID as Integer	Returns the specified resource as a string.
GetSound	ID as Integer	Returns the snd resource as a Sound object based on the *ID* passed.
ReleaseHandle	Handle as Integer	Releases a resource handle obtained from GetHandle.
RemoveResource	Type as String, ID as Integer	Removes the specified resource from the resource fork.
ResourceCount	Type as String	Returns the number of resources of the specified type.
ResourceID	Type as String, Index as Integer	Returns the resource ID as an Integer based on the *Type* and *Index* passed. Note: this list begins at zero.
ResourceName	Type as String, Index as Integer	Returns the resource name as a string based on the *Type* and *Index* passed. Note: this list begins at zero.
ResourceType	Index as Integer	Returns the resource type as a string based on the *Index* passed. Note: this list begins at zero.

Notes

With one exception, access to the resourcefork is supported only on Macintosh. Check the value of the TargetMacOS constant before attempting to access the resourcefork.

The one exception is that you can add custom cursors to a project using the CURS resource and access them from a Windows build. See the Notes for the MouseCursor class and the section "Custom Cursors in Windows Applications" on page 312 of the *User's Guide*.

In built applications, all of the fields in the Get Info area of the Build Application dialog box are saved in a standard 'vers' resource. You can access this information — such as version number, region code, and release level — using the GetResource method of the ResourceFork class.

If you use resource numbers that conflict with internal REALbasic resources, unpredictable results may occur. One way to avoid resource ID conflicts is to choose high values, like 1128 instead of 128.

An alternate way of adding custom icons to your built application is to put your icon data (including 'icns' 32-bit icons) in a resources file that you add to your project. These icons will overwrite the icon resources supplied by REALbasic itself.

More information on the resource attribute properties (ResourceLocked, ResourcePreLoad, ResourceProtected, ResourcePurgeable, ResourceSysHeap) can be found in Apple Computer's *Inside Macintosh* series.

Examples

This example sets the locked attribute of a PICT resource whose ID is 128:

```
Dim rf as ResourceFork
rf=GetFolderItem("MyApp").OpenResourceFork
rf.ResourceLocked("PICT",128)=True
```

This example checks to see if the purgable property is set:

```
Dim rf as ResourceFork
rf=GetFolderItem("MyApp").OpenResourceFork
If rf.ResourcePurgeable("PICT",128) Then
  Beep
  MsgBox "This resource is purgeable."
End If
```

This example opens the project's resource fork. If you have added a resource file to your project, you can use this technique to access your resource file (assuming that the compiled application is running on Macintosh):

```
Dim rf as ResourceFork
rf=App.ResourceFork
```

This example gets the desktop application icon for Photoshop® and displays it in a Canvas control.

```
Dim rf as ResourceFork
Dim f as FolderItem
Dim p as Picture
f=GetFolderItem("Photoshop")
If f <> Nil then
 rf=f.OpenResourceFork
 If rf <> Nil then
  p=rf.Geticl(128)
 Canvas1.backdrop=p
 Else
 MsgBox "Resource not found!"
 End if
 Else
 MsgBox "File not found!"
End if
```

This example loads a cursor from the Photoshop application and uses it as the application's cursor.

```
Dim rf as ResourceFork
Dim m as MouseCursor
Dim f as FolderItem
f=GetFolderItem("Photoshop")
If f <> Nil then
 rf=f.OpenResourceFork
 If rf <> Nil then
  m=rf.GetCursor(1525)
 App.MouseCursor=m
 Else
  MsgBox "Resource not found!"
 End if
 Else
 MsgBox "File not found!"
End if
```

See Also

App object, Window class, MouseCursor class.

Rgb Function

Returns a Color based on the red, green, and blue values specified. You can also specify a color using the HSV or CMY models.

Syntax

***result*=Rgb(*red*, *green*, *blue*)**

Part	Type	Description
result	Color	An object that represents the color based on the *red*, *green* and *blue* values.
red	Integer	The amount of red in the color (0-255).
green	Integer	The amount of green in the color (0-255).
blue	Integer	The amount of blue in the color (0-255).

Notes

The **Rgb** function returns a color object based on the amounts of red, green, and blue passed. These amounts are represented by integers between 0 and 255.

When you drag a color from the Colors palette, the dragitem includes the color's RGB values. They will be used automatically when you drag to a line in the Code Editor.

Examples

This example uses the **Rgb** function to assign various colors to the ForeColor property of a Canvas control.

```
canvas1.graphics.forecolor=Rgb(0,0,0)
//set to black
canvas1.graphics.forecolor=Rgb(255,0,0)
//set to red
canvas1.graphics.forecolor=Rgb(255,255,255)
//set to white
```

See Also

Color data type; CMY, HSV, SelectColor functions.

RGBSurface Object

Used for direct 24-bit pixel manipulations.The RGBSurface property of a Picture object allows you to manipulate the picture at the pixel level. Can be used only for pictures created by NewPicture with a pixel depth of 16 or 32.

Pixel manipulations using the RGBSurface property are faster than the same manipulations using the Graphics class methods.

Methods

Name	Parameters	Description
Pixel	x as Integer, y as Integer	Returns color of object at location specified by parameters. Returns a Color.

Note

32-bit **RGBSurface** objects are mapped to 24-bit objects on Windows.

See Also

RGBSurface property of Picture object.

Right Function

Returns the last *n* characters from the string specified.

Syntax

***result*=Right(*source*, *count*)**

OR

***result*=*stringVariable*.Right(*count*)**

Part	Type	Description
result	String	The rightmost *count* characters of *source*.
source	String	The source string from which to get the characters.
count	Integer	The number of characters you wish to get from the *source*. If *count* is greater than the length of *source*, all characters in *source* are returned.
stringVariable	String	Any variable of type String.

Notes

The **Right** function returns characters from the source string starting from the right side (as the name implies).

If you need to read bytes rather than characters, use the RightB function.

Examples

This example uses the **Right** function to return the last 5 characters from a string.

```
Dim s As String
s=Right("Hello World", 5) //returns "World"
```

See Also

Asc, Chr, InStr, Left, Len, Mid functions; String data type.

RightB Function

Returns the last *n* bytes from the string specified.

Syntax

***result*=RightB(*source*, *count*)**

OR

***result*=*stringVariable*.RightB(*count*)**

Part	Type	Description
result	String	The rightmost *count* bytes of *source*.
source	String	The source string from which to get the bytes.
count	Integer	The number of bytes you wish to get from the *source*. If *count* is greater than the length of *source*, all bytes in *source* are returned.
stringVariable	String	Any variable of type String.

Notes

The **RightB** function returns bytes from the source string starting from the right side (as the name implies).

RightB treats *source* as a series of bytes rather than a series of characters. It should be used when *source* represents binary data. If you need to read characters rather than bytes, use the Right function.

Examples

This example uses the **RightB** function to return the last 5 bytes from a string.

```
Dim s As String
s=RightB("Hello World", 5) //returns "World"
```

See Also

AscB, ChrB, InStrB, LeftB, LenB, MidB, Right functions; String data type.

Rnd function

Returns a randomly generated number between 0 and 1. Superseded by the Random class.

Syntax

***result*=Rnd**

Part	Type	Description
result	Double	A random number between 0 and 1, inclusive.

Notes

The **Rnd** function returns a number greater than zero but less than 1 with seven decimal places.

Examples

This example uses the **Rnd** function to return a random number.

```
Dim d As Double
d=Rnd
```

This example returns a random integer between 0 and 9 inclusive.

```
Dim i as Integer
i=Rnd*10
```

The following function returns a random integer between the minimum and maximum values passed to it (including minVal and maxVal).

```
Function RandInt(minVal As Integer, maxVal As Integer) As Integer
 Return minVal + Rnd * (maxVal - MinVal + 1)
End Function
```

See Also

Random class.

Round Function

Returns the value specified rounded to the nearest Integer.

Syntax

***result*=Round (*value*)**

Part	Type	Description
result	Double	The rounded *value*.
value	Double	The value you want to round.

Notes

The **Round** function returns the value passed to it rounded to the nearest Integer.

Examples

This example uses the **Round** function to return a rounded number.

```
Dim d as Double
d=Round(1.499) //returns 1
d=Round(1.500) //returns 2
```

See Also

Ceil, Floor functions.

RoundRectangle Control 

Draws a rectangle with rounded corners.

Super Class RectControl

Because this is a RectControl, see the RectControl for other properties and events that are common to all RectControl objects.

Properties

Name	Type	Description
BorderWidth	Integer	The width of the round rectangle's border in pixels.
FillColor	Color	The color of the interior of the RoundRectangle.
OvalHeight	Integer	The height of the rounded corners.
OvalWidth	Integer	The width of the rounded corners.

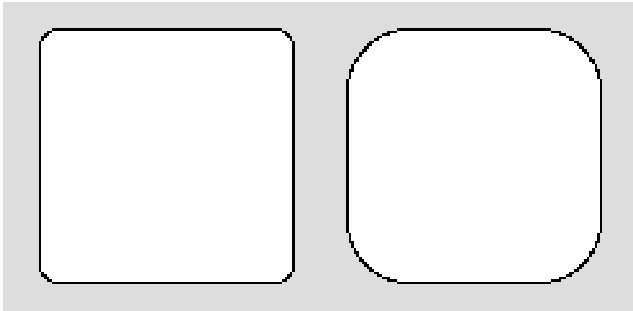
Events

Name	Parameters	Description
MouseDown	x as Integer, y as Integer	The mouse button was pressed inside the Round Rectangle region at the location passed in to x,y. Return True if you are going to handle the MouseDown.
MouseDrag	x as Integer, y as Integer	The mouse button was pressed inside the Round Rectangle and moved (dragged) at the location local to the control passed in to x,y. This event will not occur unless you return True in the MouseDown event.
MouseUp	x as Integer, y as Integer	The mouse button was released inside the Round Rectangle region at the location passed in to x,y. This event will not occur unless you return True in the MouseDown event.

Notes

Increasing the OvalWidth and OvalHeight creates rounded corners that are have a more gradual arc. In the figure below, the round rectangle on the left has an OvalWidth

and OvalHeight of 16. The round rectangle on the right has an OvalWidth and OvalHeight of 50.



Example

The following code defines a custom RoundRectangle:

```
roundrectangle1.fillColor=RGB(255,0,0) //Red
roundrectangle1.ovalheight=30
roundrectangle1.ovalwidth=16
roundrectangle1.borderwidth=1
```

See Also

Rectangle control, RectControl class.

RoundRectShape Class

Draws a (two-dimensional) rounded rectangle in a vector graphics environment.

SuperClass

RectShape

Properties

Name	Type	Description
Cornerheight	Double	Height of the inset corner oval.
Cornerwidth	Double	Width of the inset corner oval.
Segments	Integer	The number of polygon sides to use when approximation is needed. Zero means it's up to the renderer.

Example

The following method in the Paint event of a window draws a square with rounded corners.

```
Dim r as RoundRectShape
r=New RoundRectShape
r.width=75
r.height=75
r.border=100
r.bordercolor=RGB(0,0,0)
r.fillcolor=RGB(255,102,102)
r.cornerHeight=15
r.cornerWidth=15
r.borderwidth=2.5
g.DrawObject r,150,150
```

See Also

ArcShape, CurveShape, FigureShape, FolderItem, Group2D, Graphics, OvalShape, Picture, PixmapShape, RectShape, RoundRectShape, StringShape classes.

RTrim Function

Returns the string data type passed with trailing (right side) spaces removed.

Syntax

result*=RTrim*(SourceString)

OR

***result=stringVariable*.RTrim**

Part	Type	Description
result	String	*SourceString* with trailing spaces removed.
SourceString	String	The source, a copy of which, to be returned with trailing spaces removed.

Examples

This example removes the spaces from the right side of the string passed:

```
Dim s as String
s=RTrim("  Hello World  ") //Returns "  Hello World"
```

See Also

LTrim, Trim, Asc, Chr, InStr, Len, Left, Mid functions; String data type.

Runtime Object

Returns information about the current state of the runtime environment. This information can be useful for debugging purposes.

Properties

Name	Type	Description
ObjectCount	Integer	Current number of objects in existence.
ObjectClass	String	Takes ObjectNumber (Integer) as parameter, where ObjectNumber is greater than or equal to zero and less than *ObjectCount*. Returns the class of the passed object
ObjectRefs	Integer	Takes ObjectNumber (Integer) as parameter, where ObjectNumber is greater than or equal to zero and less than *ObjectCount*. Returns the number of references to the passed object.
ObjectID	Integer	Takes ObjectNumber (Integer) as parameter, where ObjectNumber is greater than or equal to zero and less than *ObjectCount*. Returns a unique identifier for the passed object.
MemoryUsed	Integer	Returns the total amount of memory used (in bytes) by the allocated objects.

Example

The following code displays the properties related to individual objects in a multicolumn ListBox and the total amount of memory used in an EditField:

```
Dim i, total as Integer
total=Runtime.ObjectCount
For i=0 to total-1
  Listbox1.addrow Str(Runtime.ObjectID(i))
  Listbox1.Cell(i,1)=Runtime.ObjectClass(i)
  Listbox1.Cell(i,2)=Str(Runtime.ObjectRefs(i))
Next
EditField1.text=Str(Runtime.memoryUsed)
```

See Also

System object.

RuntimeException Class

Super Class

Object

Properties

Name	Type	Description
Message	String	Used to contain descriptive text to display when the runtime exception is encountered.

Notes

Use runtime exceptions to trap errors so they can be handled. For example, reading from or writing to an array element that does not exist will generate an OutOfBoundsException. Exceptions have no methods or events.

Name	Description
IllegalCastException	You cast an object to a different class and sent it a message its real class can't accept.
KeyNotFoundException	An attempt was made to access a Dictionary item with a key that the dictionary does not contain.
NilObjectException	An attempt was made to access an object that does not exist.
OutOfBoundsException	An attempt was made to read from or write to a value, character, or element outside the bounds of the object or data type.
OutOfMemoryException	Raised in certain cases when an operation cannot be completed due to insufficient memory.
RbScriptAlreadyRunningException	The user tried to modify an RBscript that is already executing or tried to modify the context of the script while it is running.
RegExException	The RegEx engine issued a runtime exception. Currently this means that you used an invalid search pattern in a Regular Expression. In the future, other types of regular expression exceptions may be added.
ShellNotRunningException	You tried to access an asynchronous or interactive shell session, but the shell was not running.

Name	Description
StackOverflowException	When one routine (method/event handler/menu handler) calls another, memory is used to keep track of the place in each routine where it was called along with the values of its local variables. The purpose of this is to return (when the routine being called finishes) to the previous routine with all local variables as they were before. The memory set aside for tracking this is called the Stack (because you are "stacking" one routine on top of another). If your application runs out of stack space, a StackOverflowException will occur. You should be able to test your application thoroughly enough to prevent this error from occurring.
TypeMismatchException	You tried to assign to an object the wrong data type.
UnsupportedFormatException	You used a string expression that does not evaluate to a number or tried to open or save an unsupported picture format.

If you fail to trap a runtime error in an exception block, it will trigger the UnhandledException event of the Application class. You can then handle all runtime exceptions generically within that event handler.

If you install the Office Automation plug-ins, exception classes for each of the three plug-ins become part of the language: WordException, ExcelException, and PowerPointException. Each of these classes has two properties:

Name	Type	Description
Message	String	Indicates the command that was executed when the error occurred
Code	Integer	An OLE error code.

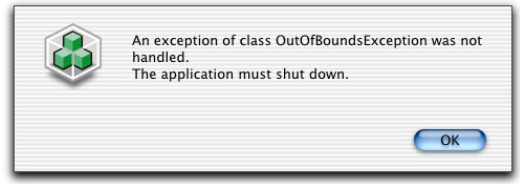
When using the Office Automation plug-ins, you can include an exception block that references these classes. For example:

```
Exception err as WordException
  MsgBox err.message+" Error No.: "+Str(err.code)
```

Third-party plug-ins may also incorporate their own types of runtime exceptions.

When you are testing your application in the IDE, execution will stop at the offending line and you will see an error message in your Code Editor. When the error occurs in a built application, the user will see a generic error message (as shown in the Notes section below) and quit — unless you include an exception handler.

Runtime Exceptions are used to trap errors in programming. Without an exception handler, the user will see a generic message from REALbasic that a particular type of exception has occured.



The application will quit when the user accepts this dialog.

Including exception handlers in your code allows you to display information that will help you determine the cause of the problem and prevents your application from quitting automatically.

Say, for example, you are getting an OutOfBoundsException when trying to access an element of an array. You could put in an exception handler to display the number of items in the array and the item number you were attempting to change along with any other useful information that would help you to track down the source of the bug. See the section on OutOfBoundsException for an example of this.

There is another example of exception handling in the section on IllegalCastException.

Examples

This example includes an If statement that checks for the type of exception and displays a message box indicating the type of exception that has occurred. The Exception block goes after the last line in the method. Note that its indentation is at the same level as the Sub statement.

```
Sub Action
 Dim a(3) as String
 a(4)="geoff"
Exception err
 If err IsA NilObjectException then
  MsgBox "Nil Object Exception"
 elseif err IsA OutOfBoundsException then
  MsgBox "Out of Bounds"
 elseif err IsA TypeMismatchException then
  MsgBox "Type Mismatch"
 End if
End Sub
```

Instead of the If statement, you can use several Exception blocks, each of which handles a specific type of runtime exception.

```
Dim f as FolderItem
f.delete
Exception err as NilObjectException
MsgBox "Nilobjectexception"
Exception err as OutOfBoundsException
MsgBox "Out of Bounds"
Exception err as TypeMismatchException
MsgBox "typemismatch"
```

The following example checks for the type of exception and handles the exception if it is a NilObjectException. If the exception is not a NilObjectException, it lets execution continue through the calling chain by calling Raise.

```
Dim f as FolderItem
f.Delete
Exception err
If err IsA NilObjectException then
 err.message="Drat!"
 MsgBox err.message
else
 Raise err
end if
```

See Also Function statement, Raise statement, NilObjectException, IllegalCastException, TypeMismatchException, OutOfBoundsException, StackOverFlowException errors; Exception block, Nil function.

SaveAsDialog Class

Used to create customized Save As dialogs. Non-customizable Save As dialogs are created by the GetSaveFolderItem function.

Super Class FolderItemDialog

Notes Using the properties of the FolderItemDialog class, you can customize the following aspects of a save-file dialog box:

- Position (Left and Top properties)
- Default directory (Initial Directory property)
- Valid file types to show (Filter property)

- Default filename (SuggestedFileName property)
- Text of the Validate and Cancel buttons (ActionButtonCaption and CancelButtonCaption properties)
- Text that appears in the Title bar of the dialog (Title property)
- Text that appears in the body of the dialog (PromptText property)

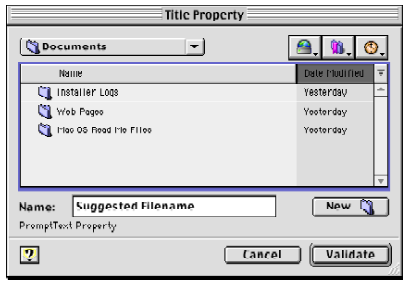
On Mac OS X, a Hide Filename Extension checkbox appears in the save-file dialog. The FolderItem returned has its ExtensionVisible property set according to the user's use of this checkbox.

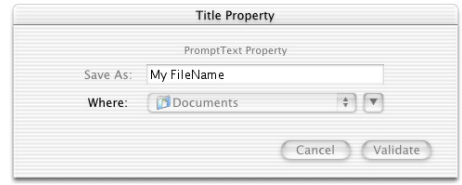
Example

The following example opens a customized save-file dialog box and displays the contents of the Mac OS 9 directory in the browser area.

```
Dim dlg as SaveAsDialog
Dim f as FolderItem
dlg=New SaveAsDialog
dlg.InitialDirectory=Volume(0).Child("Mac OS 9")
dlg.promptText="Enter a filename"
dlg.SuggestedFileName="RTF file"
dlg.Title="Save your file"
f=dlg.ShowModal()
If f <> Nil then
 //file saved
Else
//user canceled
End if
```

This is what it looks like:





See Also FolderItemDialog, OpenDialog, SelectFolderDialog, FolderItem classes; GetSaveFolderItem function.

Screen Class

Used to get information about the monitors connected to the user's computer.

Super Class Object

Properties

Name	Type	Description
AvailableHeight	Integer	The available height in pixels of the specified screen, taking into account the menubar and/or dock, if present.
AvailableLeft	Integer	The number of usable pixels from the left edge of the specified screen. This would be zero unless, for example, the user had placed the Dock on the left.
AvailableTop	Integer	The number of usable pixels from the top edge of the specified screen. It takes into account the height of the menubar, if present on the specified screen.
AvailableWidth	Integer	The available width of the screen in pixels, taking into account the Dock, if placed on the left or right of the specified screen.
Depth	Integer	The color depth of the screen.

Name	Type	Description
Height	Integer	The height of the screen.
Left	Integer	The upper-left coordinate of the screen relative to the main screen.
Top	Integer	The upper-right coordinate of the screen relative to the main screen.
Width	Integer	The width of the screen.

Notes

Although you cannot create an object of type Screen, you can access screen objects through the Screen function. See the Screen and ScreenCount functions for more information.

Example

The following reports on the values of AvailableLeft, AvailableHeight, AvailableTop, and AvailableWidth for the main screen. If the user has the Hide the Dock option on, the size of the dock when it is temporarily made visible is not taken into account.

```
Dim s as String
s="Left="+Str(Screen(0).availableleft)+Chr(13)
s=s+"Width="+Str(Screen(0).availablewidth)+Chr(13)
s=s+"Top="+Str(Screen(0).availabletop)+Chr(13)
s=s+"Height="+Str(Screen(0).availableheight)+Chr(13)
Me.text=s
```

Screen Function

Used to access the properties of a Screen object. Returns a reference to the screen passed.

Syntax

Screen(*index*).*property*

Part	Type	Description
index	Integer	The number of the screen whose property you wish to read. 0=the main screen (the screen with the menu bar).
property	Any screen object property name	The property you wish to read. See the description of the Screen class.

Notes

Although you cannot create an object of type Screen, you can use the **Screen** function to access the screen properties for the monitors attached to the user's computer. Screen 0 is the main screen (the one with the menu bar). You can use the ScreenCount function to determine how many screens exist.

Examples

This example displays the size of the user's main screen.

```
MsgBox "Your main screen is "+Str(screen(0).width)+" by " +Str(screen(0).height)+"."
```

See Also

Screen class.

ScreenCount Function

Used to determine the number of screens connected to the user's computer.

Syntax

***result*=ScreenCount**

Part	Type	Description
result	Integer	Any container expecting an Integer value.

Notes

The **ScreenCount** function returns the number of screens (monitors) connected to the user's computer.

Example

This example reports on the number of monitors attached to the user's computer.

```
Dim myscreens as Integer
myscreens=ScreenCount
If (myscreens=1) then
 MsgBox "You've only got one monitor. What's the problem?"
else
 MsgBox "You've got "+Str(myscreens)+" screens!"
end if
```

See Also

Screen class; Screen function.

Scrollbar Control

The scrollbar control.

Super Class

RectControl

Because this is a RectControl, see the RectControl for other properties and events that are common to all RectControl objects.

Properties

Name	Type	Description
LiveScroll	Boolean	If True, a ValueChanged event occurs as the user drags the thumbnail in the scrollbar. Otherwise, a single ValueChanged event occurs when the user stops dragging the thumbnail.
Maximum	Integer	The maximum value of the scrollbar. Default is 100.
Minimum	Integer	The minimum value of the scrollbar. Default is 0.
PageStep	Integer	The amount the value changes when the user clicks in the Scrollbar track.
LineStep	Integer	The amount the value changes when a scroll arrow is clicked. Default is 1
Value	Integer	The current value of the scrollbar.

Events

Name	Parameters	Description
GotFocus		(Win32 only) The scrollbar has received the focus. If the user selected the control by tabbing into it, the thumb is flashing.
LostFocus		(Win32 only) The scrollbar has lost the focus.
MouseDown	x as Integer y as Integer	The mouse button was pressed inside the Scrollbar at the location passed in to x,y. Returns a Boolean. Return True if you are going to handle the MouseDown. The coordinates x, and y are local to the control. Point 0,0 is the top-left corner of the entire control (to the left of the label and above the box that is drawn).
MouseDrag	x as Integer y as Integer	The mouse button was pressed inside the Scrollbar and moved (dragged) at the location local to the control passed in to x,y. This event will not occur unless you return True in the MouseDown event.
MouseUp	x as Integer y as Integer	The mouse button was released inside the Scrollbar at the location passed in to x,y. This event will not occur unless you return True in the MouseDown event.
ValueChanged		The scrollbar value has changed.

Note

The Controls Palette has separate tools for horizontal and vertical scrollbars. Any scrollbar can be rotated to the other orientation after it has been added to a window.

Examples

Changing the maximum value a **Scrollbar** at runtime:

```
ScrollBar1.maximum=200
```

Setting the text of staticText1 to the value of the **Scrollbar** when the user scrolls:

```
Sub ValueChanged()
  staticText1.text=Str(scrollbar1.value)
End Sub
```

See also the discussion of scrolling ListBox controls horizontally using a **Scrollbar** control.

See Also

Slider control.

Select Case Statement

Executes one of several groups of statements, depending on the value of an expression.

Syntax

Select Case *testExpression*

[Case *expression-n*

***statements-n*]**

[Else

***elseStatements*]**

End [Select]

Part	Description
testExpression	Any integer or string expression.
expression-n	Any integer or string expression.
statements-n	Statements to be executed if *expression-n* is true.
elseStatements	Statements to be executed if no expressions are True.

Notes

The **Select Case** statement is useful when there are several possible conditions that must be checked. Unlike an If statement, the **Select Case** statement will exit as soon as it finds a matching **Case** *expression* and executes any statements that follow the **Case** *expression* up to the next **Case** *expression*. If there are no **Case** *expression* that match, the *elseStatements* are executed.

Example

This example uses the **Select Case** statement to determine which day of the week the user has entered into an EditField and displays a message based on that information.

```
Dim dayNumber As Integer
Dim msg As String
dayNumber=Val(EditField1.text)
Select Case dayNumber
Case 2
  msg="It's Monday"
Case 3
  msg="It's Tuesday"
Case 4
  msg="It's Wednesday"
Case 5
  msg="It's Thursday"
Case 6
  msg="It's Friday"
Else
  msg="It's the weekend."
End Select
MsgBox msg
```

See Also

If...Then...Else statement.

Select4DDatabaseByADSP Function

Used to present an open-file dialog box for opening a 4th Dimension (4D Server) database using ADSP. Please open a 4DServer database using the ADSP4DServer (or TCPIP4DServer) classes.

Syntax

***result*=Select4DDatabaseByADSP (*task*, *username*, *password*)**

Part	Type	Description
result	Database	Database object referencing 4D database.
task	String	4D task
Username	String	Username used to log onto database.
Password	String	Password used to log onto database.

Note

To access 4D Server, it must be configured to allow 4D Open connections and the user *username* must be in the group that has 4D Open access.

To use **Select4DDatabaseByADSP**, you must have the correct 4D Server plug-in installed in your REALbasic Plugins folder. REAL Software provides separate 4D

ADSP plug-ins for version 6.0,6.5, and 6.7 of 4D Server. The REALbasic CD includes all database plug-ins and you can always find the most current versions on REAL Software's web site, www.realsoftware.com.

If you use a 4D data source on Windows, you should copy the 4DOpen.DLL file (provided by 4D) into the same directory as your application. This will ensure that the application can find this library and that the proper version of the library is used.

Example

```
Dim db as Database
db=Select4DDatabasebyADSP("mytask","Melissa","Peanuts")
If db = Nil then
 Beep
  MsgBox "The database couldn't be opened."
else
  //continue with database operations
end if
```

See Also

Open4DDatabaseByADSP, Open4DDatabasebyTCPIP, OpenCSVCursor, OpenDBFCursor, OpenDTFDatabase, OpenODBCDatabase, OpenOracleDatabase, OpenOpenBaseDatabase, OpenPostgreSQLDatabase, OpenREALDatabase, SelectODBCDatabase, Select4DDatabaseByTCPIP functions; Database, RecordSet, ADSP4DServer, TCPIP4DServer classes.

Select4DDatabaseByTCPIP Function

Used to present an open-file dialog box for opening a 4th Dimension (4D Server) database using TCP/IP. This function has been superseded by the TCPIP4DServer class.

Syntax

***result*=Select4DDatabaseByTCPIP (*IPAddress*, *task*, *username*, *password*)**

Part	Type	Description
result	Database	Database object referencing 4D database.
IPAddress	String	IP address of server machine
task	String	4D task
Username	String	Username used to log onto database.
Password	String	Password used to log onto database.

Note

To access 4D Server, it must be configured to allow 4D Open connections and the user *username* must be in the group that has 4D Open access.

To use **Select4DDatabaseByTCPIP**, you must have the correct 4D Server plug-in installed in your REALbasic Plugins folder. REAL Software provides separate 4D ADSP plug-ins for version 6.0, 6.5, and 6.7 of 4D Server. The REALbasic CD includes all database plug-ins and you can always find the most current versions on REAL Software's web site, www.realsoftware.com.

If you use a 4D data source on Windows, you should copy the 4DOpen.DLL file (provided by 4D) into the same directory as your application. This will ensure that the application can find this library and that the proper version of the library is used.

Example

```
Dim db as Database
db=Select4DDatabasebyTCPIP("192.168.1.5","mytask","Melissa","Peanuts")
If db = Nil then
 Beep
  MsgBox "The database couldn't be opened."
else
  //continue with database operations
end if
```

See Also

Open4DDatabaseByADSP, Open4DDatabasebyTCPIP, OpenCSVCursor, OpenDBFCursor, OpenDTFDatabase, OpenODBCDatabase, OpenOracleDatabase, OpenOpenBaseDatabase, OpenPostgreSQLDatabase, OpenREALDatabase, SelectODBCDatabase functions; Database, RecordSet, ADSP4DServer, TCPIP4DServer classes.

SelectColor Function

Displays the standard Color picker, allowing the user to choose a color.

Syntax

***result*=SelectColor(ByRef *col, prompt*)**

Part	Type	Description
result	Boolean	Returns True if the user clicks OK; False if the user cancels.
col	Color	Color passed to SelectColor and the color the user selects. Note usage of ByRef in the parameter list.
prompt	String	Prompt to be displayed in the Color PIcker.

Notes

You control the default choice of color displayed by the Color Picker by passing a value of *col* to **SelectColor**. If *col* is not assigned a value, black is used as the default color. The color that the user chooses is returned in *col*. If the user clicks Cancel without making a choice, *result* is set to False; otherwise it is set to True.

Example

The following example allows the user to select a color that will be used as the fillcolor in a Rectangle control.

```
Dim c  as Color
Dim b as Boolean
c=CMY(.35,.9,.6) //choose the default color shown in color picker
b=SelectColor(c,"Select a Color")
Rectangle1.FillColor=c
```

See Also

Color data type; CMY, RGB, HSV functions.

SelectFolder Function

Displays an open-file dialog box allowing the user to select a folder rather than a file.

Syntax

***result*=SelectFolder**

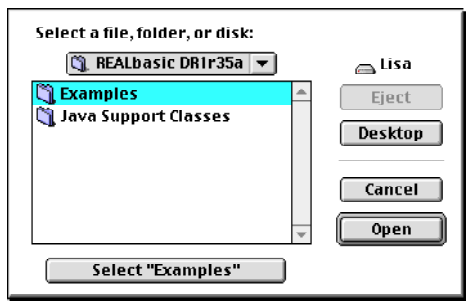
Part	Type	Description
result	FolderItem	The folder the user selected.

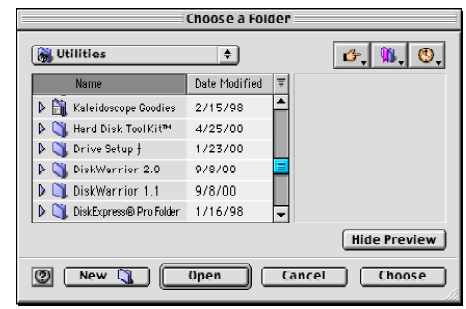
Notes

The **SelectFolder** function displays a dialog box similar to the open-file dialog box displayed by the GetOpenFolderItem function. The difference is that the dialog box displayed by the **SelectFolder** function allows the user to choose a folder rather than a file.

If the user has Mac OS System 8.5 (or above) installed, the new Navigation Services browser is displayed as a movable modal dialog box. It is shown on the right in the following illustration.

Original and Navigation Services open-folder dialog boxes.





Examples

This example displays the name of the folder the user chose.

```
Dim f as FolderItem
f=SelectFolder
If f<> Nil Then
  MsgBox f.name
End If
```

See Also

GetOpenFolderItem, GetSaveFolderItem functions.

SelectFolderDialog Class

Used to create and present customized Select Folder dialog boxes. The non-customized version of this function is provided by the SelectFolder function.

Super Class

FolderItemDialog class.

Notes

Using the properties of the FolderItemDialog class, you can customize the following aspects of a select-folder dialog box:

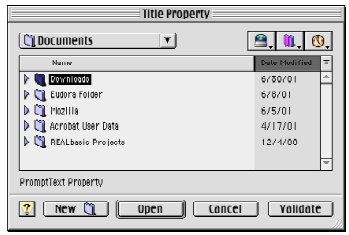
- Position (Left and Top properties)
- Default directory (Initial Directory property)
- Valid file types to show (Filter property)
- Text of Validate and Cancel buttons (ActionButtonCaption and CancelButtonCaption properties).
- Text that appears in the Title bar of the dialog (Title property)
- Text that appears in the body of the dialog (PromptText property)

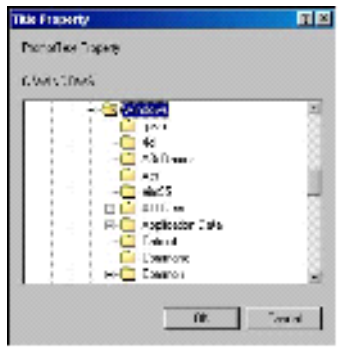
Example

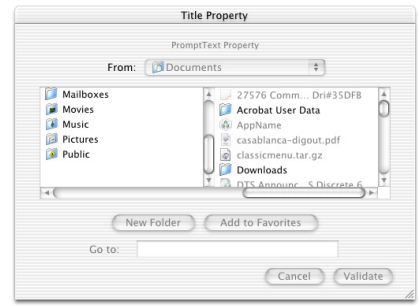
The following example opens a select folder dialog box and presents the contents of the Mac OS 9 folder on the user's startup volume in the browser:

```
Dim dlg as SelectFolderDialog
Dim f as FolderItem
dlg=New SelectFolderDialog
dlg.ActionButtonCaption="Select"
dlg.InitialDirectory=Volume(0).Child("Mac OS 9")
f=dlg.ShowModal()
if f <> Nil then
 //use the folderitem here
else
 //user cancelled
end if
```

This is what it looks like:







See Also

FolderItemDialog, OpenDialog, SaveAsDialog, FolderItem classes; SelectFolder function.

SelectODBCDatabase Function

Allows the user to open an ODBC database. When SelectODBCDatabase is called, the user is presented with an open-file dialog box that allows the user to open any available ODBC database.

When you use **SelectODBCDatabase**, it will prompt you to choose either a File DSN (Data Source Name) or a User DSN.

Syntax

***result*=SelectODBCDatabase**

Part	Type	Description
result	Database	Reference to ODBC database

Notes

SelectODBCDatabase requires the ODBC Database plug-in. Place this plug-in in the REALbasic Plugins folder. The REALbasic CD includes all database plug-ins and you can always find the most current versions on REAL Software's web site, www.realsoftware.com.

SelectODBCDatabase also requires a third-party driver manager and drivers. REALbasic supports OpenLink's ODBC Driver manager and drivers (www.openlinksw.com) and Merant/Metro Technologies ODBC Manager. OpenLink supports both Mac OS 8/9 and Mac OS X, while Merant currently does not support Mac OS X. A driver manager must be installed as well as the driver for whichever database back end you wish to access. Check REAL Software's web site, www.realsoftware.com, for updated drivers.

Example

```
Dim DB as Database
DB=SelectODBCDatabase
If db = Nil then
  Beep
  MsgBox "The database couldn't be opened."
else
  //continue with database operations
end if
```

See Also

Open4DDatabaseByADSP, Open4DDatabasebyTCPIP, OpenCSVCursor, OpenDBFCursor, OpenDTFDatabase, OpenODBCDatabase, OpenOracleDatabase, OpenOpenBaseDatabase, OpenPostgreSQLDatabase, OpenREALDatabase, Select4DDatabaseByADSP, Select4DDatabaseByTCPIP, functions; Database, RecordSet ODBCDatabase classes.

Self Function

Self is a reference to an object's super (parent) object.

Notes

When called within the method of an object on a form, **Self** will always be a reference to the object's parent window. This means that in a PushButton control's Action event handler:

Left=100 and **Self**.Left=100

mean the same thing because, in the absence of an object identifier, the parent object (in this case, the window the PushButton in a part of) is assumed.

For code for a control in a window, "Self" refers to the window, and "Me" refers to the control.

When **Self** is called within the method of a class, **Self** will be a reference to the instance of the class in use. This is usually not necessary since, as stated, in the absence of an object identifier, the parent or super object is assumed.

See Also

Me function.

Self Cannot be used in the Method of a Module Error

You can't use Self or Me in a method or function in a module because there is no parent window or control.

See Also

Self function.

Semaphore Class

Used to control access to shared code in a multithreaded environment.

Super Class

Object

Methods

Name	Parameters	Description
Signal		Call to acquire ownership of a semaphore.
Release		Call to release access to a semaphore.

Notes

A semaphore is an object that can be used to synchronize access to a shared resource.

To acquire the ownership of a semaphore, a thread would call the Signal method. If the semaphore isn't owned, the thread acquires ownership, otherwise the thread will be forced to wait until the Semaphore is released via the Release method.

See Also

CriticalSection, Thread classes.

Separator Control

Used to add a separator to a window.

Super Class

RectControl

Because this is a RectControl, see the RectControl for other properties and events that are common to all RectControl objects.

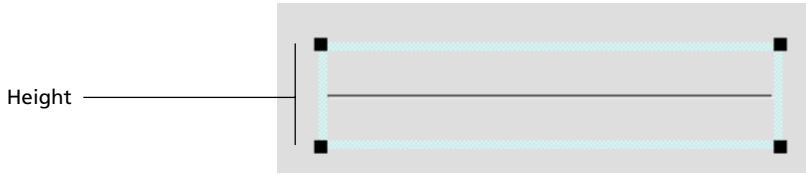
Properties

Name	Type	Description
Height	Integer	The height of the separator.
Left	Integer	The distance in pixels from the left of the window to the left of the separator.
LockBottom	Boolean	Determines whether the bottom edge of the control should stay at a set distance from the bottom edge of the owning window.
LockLeft	Boolean	Determines whether the left edge of the control should stay at a set distance from the left edge of the owning window. *LockLeft* has no effect unless *LockRight* is True.
LockRight	Boolean	Determines whether the right edge of the control should stay at a set distance from the right edge of the owning window.
LockTop	Boolean	Determines whether the top edge of the control should stay at a set distance from the top edge of the owning window. *LockTop* has no effect unless *LockBottom* is True.
Top	Integer	The distance in pixels from the top of the window to the top of the separator.
Width	Integer	The width of the separator.

Notes

If the width is greater than the height, the separator is horizontal; otherwise it is vertical. In either case, the smaller value controls the thickness of the Separator control, not the thickness of the visible line in the center of the control. Mouse movement events are detected over the entire area of the Separator control, not just the visible line.

The following illustration shows a very thick separator.



See Also

RectControl class.

Serial Control

Serial controls are used to perform serial communications.

Super Class

Object

The Serial control can be instantiated via code since it is not a subclass of Control. This allows you to easily write code that does communications without adding the control to a window.

Properties

Name	Type	Description
Baud	Integer	The rate at which data will be sent and received. (see chart below)
Bits	Integer	0=5 data bits, 1=6 data bits, 2=7 data bits, 3=8 data bits
CTS	Boolean	Enables CTS flow control.
DTR	Boolean	Enables DTR flow control.
Index	Integer	The number of the control when it's part of a control array.
Left	Integer	The left side of the control in local coordinates (relative to the window).
MacInDriverRefNumber	Integer	Macintosh In Driver Reference Number. Of use to toolbox programmers.
MacOutDriverRefNum	Integer	Macintosh Out Driver Reference Number. Of use to toolbox programmers.
Name	String	The name of the object.

Name	Type	Description
Parity	Integer	0=no parity, 1=odd parity, 2=even parity
Port	Integer	The port used to send and receive data. 0=modem port, 1=printer port.
SerialPort	SerialPort	Used to identify the serial port to which the Serial control will communicate
Stop	Integer	0=1 stop bits, 1=1.5 stop bits, 2=2 stop bits
Super	Object Class	The class of object the object is based on.
Top	Integer	The top of the control in local coordinates (relative to the window).
Win32DriverHandle	Integer	Windows Driver Handle.
XON	Boolean	Enables XON flow control.

Events

Name	Parameters	Description
DataAvailable		New incoming data is now available in the buffer.

Methods

Name	Parameters	Description
Close		Closes the serial port.
Flush		Clears all data from the serial port buffer.
LeaveDTROnClose		Tells the serial control not to negate DTR on close. The serial port must be open for this method to function.
LookAhead	[Encoding as TextEncoding]	Returns a String. Returns all the unread characters in the buffer without deleting them from the buffer.Optional *Encoding* parameter enables you to specify the text encoding of the data to be returned.
Open		Attempts to open the serial port. Returns True if the port was opened and False if it was not.
Poll		Causes the control's properties to update and causes the DataAvailable event to execute if any new data is available.
Read	Bytes as Integer, [Encoding as TextEncoding]	Reads *Bytes* of data from the buffer and returns them as a string. The optional *Encoding* parameter enables you to specify the text encoding of the data to be read.

Name	Parameters	Description
ReadAll	[Encoding as TextEncoding]	Returns all incoming data available in the buffer. The optional *Encoding* parameter enables you to specify the text encoding of the data to be read.
Reset		Resets the port's baud and byte format. The serial port must be open for this method to function.
Write	Data as String	Asynchronously writes *Data* to the serial port.
XmitWait		Waits until all data sent to the serial port with the Write method has been sent.

Baud Rates

Baud Rate	Value	Baud Rate	Value
300	0	7200	7
600	1	9600	8
1200	2	14400	9
1800	3	19200	10
2400	4	28800	11
3600	5	38400	12
4800	6	57600	13

Notes

The Serial control can be used to communicate via multiple serial ports at once. You can use the properties of the System object to determine the number of serial ports on the computer and get an array of those ports.

When data is received by a serial port connection, the DataAvailable event handler will automatically execute. In this event handler, you would then use the Read or ReadAll functions to access the data in the serial port buffer. These functions remove the data from the serial port buffer as they return the data. If you need to read the data from the serial port buffer without removing it from the buffer, use the LookAhead property. This buffer will use as much memory as it needs from the memory available so there is no need for it to be resized.

Because the Write method is handled asynchronously, you may need to use the XmitWait method to force REALbasic to wait until REALbasic has finished sending the data out the serial port.

The Serial ports are available for use under Mac OS X. Use the System object's serialport property to access the serial ports, as usual.

Example

The following code directs the Serial control to communicate using Serial port zero (Modem port) on Macintosh.

```
If TargetMacOS then //app is running on Mac
 Serial.SerialPort=system.SerialPort(0)
end if
```

See Also

SerialPort, Object classes; SystemClass object.

SerialPort Class

Used to get the properties of any serial port.

Super Class

Object

Properties

Name	Type	Description
Name	String	Name of serial port. On Macintosh, serial port zero is the Modem port and serial port 1 is the Printer port.
InputDriverName	String	Name of input driver.
OutputDriverName	String	Name of output driver.
RatedSpeed	Integer	Rated speed (baud)
MaximumSpeed	Integer	Maximum speed (baud)

Example

See the example for the System object. You get the values of serialport properties by first creating a system object.

See Also

System object.

ServerSocket Class

Used to support multiple connections on the same port. The **ServerSocket** is available only in the Professional version of REALbasic.

Super Class Object

Properties

Name	Type	Description
IsListening	Boolean	True when the **ServerSocket** is listening for incoming connections.
LocalAddress	String	The local address that the **ServerSocket** is using.
MaximumSocketsConnected	Integer	The maximum number of connections the **ServerSocket** will allow to connect. When a socket disconnects from the **ServerSocket**, the server will allow one more connection.
MinimumSocketsAvailable	Integer	The smallest number of sockets available in the server's pool of socket connections. If the **ServerSocket** falls below this number, it will call the AddSocket event to replenish its supply of sockets.
Port	Integer	The port the **ServerSocket** is listening on.

Events

Name	Parameters	Description
AddSocket		The **ServerSocket** is requesting that you add a socket to its internal pool. It is called when the **ServerSocket** first begins listening. Initially it will be called *MaximumSocketsConnected* times to fill its internal pool of sockets. This socket must stay resident (cannot be a local variable) and must be non-Nil. Returns a TCPSocket. Since UDP is a connectionless protocol, it does not make sense for a **ServerSocket** to deal with UDPSockets.
Error	ErrorCode as Integer	Occurs when the socket has an error. Standard error codes are returned. On Macintosh, they are documented in MacErrors.h, and for Windows error codes, in WinSock.h.

Methods

Name	Parameters	Description
Listen		Begins the listening process for the **ServerSocket** with the properties you specified.
StopListening		Terminates listening on the **ServerSocket**. Does not terminate any established connections.

Notes

A **ServerSocket** is a permanent socket that listens on a single port for multiple connections. When a connection attempt is made on that port, the **ServerSocket** hands the connection off to another socket, and continues listening on the same port. Without the **ServerSocket**, it is difficult to implement this functionality due to the latency between a connection coming in, being handed off, creating a new listening socket, and restarting the listening process. If you had two connections coming in at about the same time, one of the connections may be dropped because there was no listening socket available on that port.

You can change the MinimumSocketsAvailable and MaximumSocketsConnected properties after establishing the listening socket. If you change the MaximumSocketsConnected property, it will not kill any existing connections (it just may not allow more connections until the existing connections have been released). If you change the MinimumSocketsAvailable property, it may fire the AddSocket event to replenish its internal buffer.

On Mac OS X, a **ServerSocket** cannot access ports below 1024 unless the user has root privileges. This is part of the design of Mac OS X.

The **ServerSocket** and SSLSocket are available only in the Professional version of REALbasic.

See Also

UDPSocket, SSLSocket, SocketCore classes.

Shell Class

Used to execute Unix or DOS shell commands under Mac OS X or Windows, respectively.

Super Class

Object

Properties

Name	Type	Description
ErrorCode	Integer	Returns 0 if the Execute method was executed successfully. Otherwise, returns a system-supplied error code. On Windows, returns -1 if execution fails.
IsRunning	Boolean	True if an asynchronous or interactive shell process is running.

Name	Type	Description
Mode	Integer	Controls the operation of the shell. Asynchronous and Interactive modes are supported only on Mac OS X. 0=Synchronous. The shell executes its command and returns the result. 1=Asynchronous. The shell executes its command and returns data via the DataAvailable event. 2=Interactive. Data can be sent to a running shell session with the Write method and data is returned via the DataAvailable event.
PID	Integer	ID of the interactive shell process
Result	String	Contains the contents of output buffer if you are in Synchronous mode (mode 0) without clearing the buffer.
TimeOut	Integer	(Windows only) Time (milliseconds) that specifies how long a process can run before it is automatically terminated. Default is 2000 or 2 seconds.

Events

Name	Parameters	Description
Completed		Fired only in modes 1 and 2. Triggered whenever the executed command is finished.
DataAvailable		Fired only in modes 1 and 2. Triggered whenever the Shell object returns data— just like the Socket class.

Methods

Name	Parameters	Description
Close		Shuts down the running command.
Execute	Command as String	Executes a one-line Unix or DOS shell command. Used in Synchronous mode. If you are in Synchronous mode (mode 0), the Result property will contain the results.
Poll		Looks for data returned from the running shell and may trigger a DataAvailable event.
ReadAll		Returns the contents of the output buffer and clears the buffer.
Write	Text as String	Sends a string to the shell's input buffer.
WriteLIne	Text as String	Sends a string ending in a linefeed to the shell's input buffer.

Notes

Use the **Shell** class to execute DOS or Unix commands and get the results. It has a single method, Execute, which executes a one-line command. This causes two properties of the **Shell** object to change: ErrorCode, which is a system-supplied error code or 0 for no error; and Result, which is a string containing the output of the command. The TimeOut property specifies how long (in milliseconds) a process can run before it is automatically terminated. A value of -1 means the process can run

indefinitely. This property currently only applies to Windows (not needed on Mac OS X).

For Mac OS X only, the Shell class can execute commands in Asynchronous or Interactive modes.

Examples

Using the synchronous mode, the following code lists the current directory's files (Mac OS X).

```
Dim s As Shell
s=New Shell
If TargetCarbon then
  s.execute "ls -la"
  If s.errorCode=0 then
    EditField1.text=s.result
  else
    MsgBox "Error Code: "+Str(s.errorCode)
  end if
end if
```

The following terminal application allows you to submit Unix commands using the interactive mode. The interface consists of two EditFields, InputField, in which the user can enter a command, and OutputField that displays the results.

The Open event for the window initializes the shell object (declared as a property of the window).

```
mShell = New Shell
mShell.Mode = 2
```

The user can type a unix command into the EditField. When he presses Return, the following code in the EditField's Keydown event runs. The Write method sends the command to the Shell's input buffer.'

```
If Key = Chr(13) Then
 If Not mShell.IsRunning Then
  mShell.Execute "sh"
 End If
 mShell.Write InputField.Text
 mShell.Write Chr(13)
 InputField.Text = ""
 Return True
Else
 Return False
End If
```

A Timer control calls the ReadAll method in its Action event:

```
If mShell <> Nil Then
 OutputField.SelText = mShell.ReadAll
End If
```

See Also TargetCarbon, TargetWin32 constants.

ShellNotRunningException Error

You tried to pass a string to a Shell that is not running.

Super Class RuntimeException

Notes The **ShellNotRunningException** error will occur if you try to pass a string for the Shell to execute but it is not running. This can happen if you are using the Shell interactively and it has been closed prior to passing the string.

See Also Shell class.

ShowURL Method

Uses the appropriate application (based on the user's Internet Config settings) to go to the URL (web address, ftp address, etc.) specified.

Syntax **ShowURL *URL***

Part	Type	Description
URL	String	The URL to go to.

Notes URL stands for Universal Resource Locator. URL's are addresses for Internet locations such as ftp and web sites. The **ShowURL** method uses the user's Internet Config application preferences to determine which application to launch based on the type of URL specified. For example, passing a URL that beings with "http" will launch the user's web browser. Some browsers have limits on the length of a URL.

Examples This example will launch the user's chosen web browser and go to our web page.

```
ShowURL "http://www.realsoftware.com"
```

This example will launch the user's chosen ftp client application and go to our ftp site, specifically, to our current release directory.

```
ShowURL "ftp://ftp.realsoftware.com/current_release"
```

This example will launch the user's chosen e-mail client application and create a new e-mail message, with the recipient's e-mail address in the "To" area:

```
ShowURL "mailto:batman@batcave.org"
```

ShutDownItemsFolder Function

Used to access the ShutDown Items folder in the System folder.

Syntax

***result*=ShutDownItemsFolder**

Part	Type	Description
result	FolderItem	A FolderItem that represents the ShutDown Items folder.

Notes

Use the **ShutDownItemsFolder** function to access the Shut Down Items folder.

The **ShutDownItemsFolder** function provides a way to access the Shut Down Items folder that will work under different language systems, and will continue to work even if Apple reorganizes the System folder.

If your application will run on Windows, check the value of TargetMacOS or TargetWin32 before calling ShutDownItemsFolder to determine the current operating system.

Examples

This example places in a ListBox all the names of the items in the Shut Down Items folder.

```
Dim i as Integer
Dim f as FolderItem
f=ShutDownItemsFolder
for i=1 to f.count
  ListBox1.addrow f.item(i).name
next
```

See Also

ControlPanelsFolder, DesktopFolder, ExtensionsFolder, FontsFolder, PreferencesFolder, StartupItemsFolder, SystemFolder, TemporaryFolder, TrashFolder functions.

Sign Function

Returns the sign of the number passed to it.

Syntax

***result*=Sign(*value*)**

Part	Type	Description
result	Integer	The sign of *value*. Returns- 1 if *value* is negative, 0 if *value* is zero, and 1 if *value* is positive.
value	Double	The number being passed to the function.

Example

The following example determines the sign of the number passed to it.

```
Dim d as Double
Dim result as Integer
If EditField1.text <>"" then
 result=Sign(Val(EditField1.text))
 MsgBox Str(result)
Else
 MsgBox "Please enter a number!"
End if
```

See Also

Abs, Val functions.

Sin Function

Returns the sine of the value specified.

Syntax

***result*=Sin (*value*)**

Part	Type	Description
value	Double	The value you want the sine of.
result	Double	The sine of *value*.

Notes

The **Sin** function returns the sine of the angle (in radians) passed to it. If value is in degrees, multiply it by PI/180 to convert it to radians.

Examples

This example uses the **Sin** function to return the sine of a number.

```
Dim d As Double
Const PI=3.14159265358979323846264338327950
d=Sin(0.5) //returns 0.4794255
d=Sin(30*PI/180) //returns .5
```

See Also Asin function.

Single Data Type

A **Single** is an intrinsic data type in REALbasic. A **Single** is a number that can contain a decimal value, i.e., a real number. It can take on a value between -1.175494 e-38 and 3.402823 e+38. In other languages, REALbasic's **Single** may be referred to as a single precision real number. Because **Singles** are numbers, you can perform mathematical calculations on them. **Singles** use 4 bytes of memory.

The VarType function returns a value of 5 when passed a **Single**.

Example The **Single** and Double data types allow you to store and manage floating point numbers.

```
Dim s as Single
s=3.1416
```

See Also Integer, Double, String, Boolean data types; -, +, *, /, <, <=, =, >=, >, <>, \, Mod, Val, Str, VarType functions.

Slider Control

Implements the slider control on Macintosh computers running Mac OS 8.x or above. If the user is running an earlier Macintosh system, slider controls appear as Scrollbars.

Super Class RectControl

Because this is a RectControl, see the RectControl for other properties and events that are common to all RectControl objects.

Properties

Name	Type	Description
LiveScroll	Boolean	If True, a ValueChanged event occurs as the user drags the thumbnail in the slider. Otherwise, a single ValueChanged event occurs when the user stops dragging the thumbnail.
Maximum	Integer	The maximum value of the slider. The default is 100.
Minimum	Integer	The minimum value of the slider.
PageStep	Integer	The amount the value changes when the user clicks in the Slider track.

Name	Type	Description
LineStep	Integer	*LineStep* doesn't do anything on appearance manager savvy Mac-apps, but on Windows it is the amount a slider moves when the slider has the focus and the user uses the arrow keys.
Value	Integer	The current value of the slider.

Events

Name	Parameters	Description
GotFocus		(Win32 only) The slider has received the focus and has a selection marquee.
IsConnected		Returns a Boolean. Returns True if the Socket has a live connection. (Win32 only)
LostFocus		(Win32 only) The slider has lost the focus.
ValueChanged		The slider value has changed.
MouseDown	x as Integer y as Integer	The mouse button was pressed inside the Slider at the location passed in to x,y. Returns a Boolean. Return True if you are going to handle the MouseDown. The coordinates x, and y are local to the control. Point 0,0 is the top-left corner of the entire control (to the left of the label and above the box that is drawn).
MouseDrag	x as Integer y as Integer	The mouse button was pressed inside the Slider and moved (dragged) at the location local to the control passed in to x,y. This event will not occur unless you return True in the MouseDown event.
MouseUp	x as Integer y as Integer	The mouse button was released inside the Slider at the location passed in to x,y. This event will not occur unless you return True in the MouseDown event.

Examples

Changing the maximum value of a **Slider** at runtime:

```
Slider1.maximum=200
```

Setting the text of staticText1 to the value of the slider when the user scrolls:

```
Sub ValueChanged()
StaticText1.text=Str(Slider1.value)
End Sub
```

See Also

RectControl class; Scrollbar control.

SMTPSecureSocket Class

Used to send secure email via the SMTP protocol using SSL or TLS encryption.

Super Class

SSLSocket

Properties

Name	Type	Description
Messages	EmailMessage	The queue of messages that are waiting to be sent. This is an array of which the 0^{th} element is the next message to be sent. When a message is sent, it is removed from the queue and passed to the MessageSent event. To append a message to the queue, use the Append method. To remove a message, use the Remove method.

Methods

Name	Parameters	Description
DeleteAllMessages		Deletes all messages from the send queue.
DisconnectFromServer		Closes the connection with the mail server.
SendMail		Connects to the mail server and sends the messages in the queue.

Events

Name	Parameters	Description
ConnectionEstablished	Greeting as String	Executes when the connection has been established with the mail server and passes the greeting string that the server returned.
Error		An error has occurred.
MailSent		Executes when all of the messages in the queue have been sent.
MessageSent	Email as EmailMessage	Executes when a message has been sent; *Email* contains the message that was sent.
ServerError	ErrorID as Integer, ErrorMessage as String, Email as EmailMessage	Executes when an error occured while sending a message. *ErrorID* and *ErrorMessage* are sent by the mail server and *Email* is the message that was being sent when the error occurred. *Email* is removed from the queue at the time the event executes.

Notes

The **SMTPSecureSocket** class is the same as the SMTPSocket class except that it is derived from SSLSocket instead of TCPSocket. As a result, you can use the Secure property of the SSLSocket class to provide secure communications.

See Also

POP3Socket, POP3SecureSocket, EmailMessage, TCPSocket, HTTPSocket, HTTPSecureSocket, SMTPSocket, SocketCore classes.

SMTPSocket Class

Used to send email via the SMTP protocol.

Super Class

TCPSocket

Properties

Name	Type	Description
Messages	EmailMessage	The queue of messages that are waiting to be sent. This is an array of which the 0^{th} element is the next message to be sent. When a message is sent, it is removed from the queue and passed to the MessageSent event. To append a message to the queue, use the Append method. To remove a message, use the Remove method.

Methods

Name	Parameters	Description
DeleteAllMessages		Deletes all messages from the send queue.
DisconnectFromServer		Closes the connection with the mail server.
SendMail		Connects to the mail server and sends the messages in the queue.

Events

Name	Parameters	Description
ConnectionEstablished	Greeting as String	Executes when the connection has been established with the mail server and passes the greeting string that the server returned.
Error		An error occurred.
MailSent		Executes when all of the messages in the queue have been sent.
MessageSent	Email as EmailMessage	Executes when a message has been sent; *Email* contains the messages that was sent.

Name	Parameters	Description
SendComplete	UserAborted as Boolean	All of the data in the socket buffer has been written (sent). The *UserAborted* parameter enables you to determine whether the user cancelled the transfer by returning True from the SendProgress event. If the user aborted, this value is True, otherwise it is False.
SendProgress	BytesSent as Integer BytesLeft as Integer	Occurs when your network provider queues your data in 'chunks' and is about to send the next chunk. The parameters indicate the amount of progress that has been made during the send. Returns a Boolean. Returning True from this event causes the send to be aborted.
ServerError	ErrorID as Integer, ErrorMessage as String, Email as EmailMessage	Executes when an error occured while sending a message. *ErrorID* and *ErrorMessage* are sent by the mail server and *Email* is the message that was being sent when the error occurred. *Email* is removed from the queue at the time the event executes.

Notes

The POP3Socket and **SMTPSocket** classes are used together to form the basis of an email client. POP3 is the standard internet protocol for receiving messages and SMTP (Simple Mail Transfer Protocol) is the standard internet protocol for sending emails.

Example

The following code in the Action event of a PushButton sends an email message. It works with a form with fields for each part of the email. EditFields are used for the Username and Password of the email account, the From, To, and CC addresses, the subject of the email, and the body in plain text and HTML form. Since the user can enter multiple To and CC addresses, the example parses theses strings and adds the individual To and CC addresses to the EmailMessage using the AddRecipient and AddCCRecipient methods.

An attachment can be specified in the form of a full path to the file to be attached. If an attachment exists, it is added to the EmailMessage's Attachments property.

An SMTPSocket named SMTPSocket1, has been added to the window.

```
Dim mail as EmailMessage
Dim file as EmailAttachment
Dim i as Integer
Dim s as String

SMTPSocket1.address = "mail.mySMTPServer.com"
SMTPSocket1.port = 25

SMTPSocket1.username = usernameFld.text  //get username from editfield
SMTPSocket1.password = passwordFld.text  //get password from Editfield

mail = New EmailMessage
mail.fromAddress=fromAddressFld.text  //get From address from EditField
mail.subject=subjectFld.text  //get Subject of mail from EditField
mail.bodyPlainText = bodyFld.text   //get body in plain text from EditField
mail.bodyHTML = htmlFld.text  //get body in HTML from EditField
mail.headers.appendHeader "X-Mailer","REALbasic SMTP Demo"

//multiple To addresses separated by commas
//each to address added to Recipient array
s = ReplaceAll(toAddressFld.text,",",Chr(13))
For i = 1 to CountFields(s,Chr(13))
 mail.addRecipient Trim(NthField(s,Chr(13),i))
next

//multiple CC addresses separated by commas
//each to address added to CCRecipient array

s = ReplaceAll(ccAddressFld.text,",",Chr(13))
For i = 1 to CountFields(s,Chr(13))
 mail.addCCRecipient Trim(NthField(s,Chr(13),i))
next

If fileFld.text <> "" then //is there a path to an attachment file?
 file = New EmailAttachment
 file.loadFromFile GetFolderItem(fileFld.text)
 mail.attachments.append file //add file to attachments array
end

SMTPSocket1.messages.append mail //add email to list of messages
SMTPSocket1.SendMail //send message
```

See Also

POP3Socket,POP3SecureSocket, EmailMessage, EmailAttachment, EmailHeader, TCPSocket, HTTPSocket, HTTPSecureSocket, SMTPSecureSocket, SocketCore classes.

Socket Class

Socket controls are used to perform communications via TCP/IP. This class is superseded by the TCPSocket class. The **Socket** control in the Controls palette now works as a TCPSocket control.

Super Class

TCPSocket

Properties

Name	Type	Description
Address	String	The IP address to send data to.
BytesAvailable	Integer	The number of bytes of data available in the internal receive buffer.
IsConnected	Boolean	Indicates whether the socket has a live connection.
LastErrorCode	Integer	The last error that occured. The control now retains the LastErrorCode information, even after leaving the Error event.
LocalAddress	String	**The IP address of the user's computer.**
Port	Integer	The TCP/IP port to transmit/receive data on.
PPPStatus	Integer	This property is superseded by the PPPStatus property of the System object.
RemoteAddress	String	Returns the address of the machine connected to the socket. Valid only after the connection is established.

Events

Name	Parameters	Description
Connected		A connection has been established using the Address and Port properties.
DataAvailable		New incoming data is now available in the buffer.
Error		An error occured.
SendComplete	UserAborted as Boolean	All of the data in the socket buffer has been written (sent). The *UserAborted* parameter enables you to determine whether the user cancelled the transfer by returning True from the SendProgress event. If the user aborted, this value is True, otherwise it is False.

Name	Parameters	Description
SendProgress	BytesSent as Integer BytesLeft as Integer	Allows you to determine send speeds, and tells you how many bytes of data you have sent since your last SendProgress event, as well as how many bytes are left to send. Returns a Boolean. Return True from the SendProgress event to cancel the current transfer. This does not close the socket's connection, it only clears the send buffer. You can use this event to determine that a connection is too slow, and cancel it. Once all of the data has been transferred, you will get a last SendProgress event, followed by a SendComplete event.

Methods

Name	Parameters	Description
Close		Closes the connection.
Connect		Makes a connection using the Address and Port properties. May or may not successfully establish the connection. Use the IsConnected property or the Connected event to determine whether the connection attempt is successful.
Disconnect		Disconnects the socket, resets it, and issues a Socket 102 error to let you know that the socket has been disconnected.
Listen		Waits for a connection on the port specified in the Port property.
LookAhead		Returns all the unread characters in the buffer without deleting them from the buffer.
Poll		Causes the control's properties to update and causes the DataAvailable event to execute if any new data is available.
PPPConnect		Connects a dialup connection. Superseded by the PPPConnect method of the System object.
PPPDisconnect		Disconnects a PPP connection. Superseded by the PPPDisconnect method of the System object.
Purge		Removes all data from the socket's internal receive buffer. It does not affect the socket's internal send buffer.
Read	Count as Integer	Returns up to *Count* characters as a string from the socket control buffer.
ReadAll		Returns all the text in the socket control buffer.
Write	Text as String	After a connection had been established, this method sends the *Text* to the remote machine to which a connection is established.

Notes

The **Socket** class has been replaced with the TCPSocket class. In the Controls palette, the icon previously used for the Socket control is now used for the TCPSocket control.

The Socket class still functions as in previous version. We highly recommend that you use TCPSocket instead because it is much faster.

The PPPConnect and PPPDisconnect methods and the PPPStatus properties are superseded by the System object's PPPConnect, PPPDisconnect methods and PPPStatus properties. This is because it really doesn't make much sense. A PPP connection (point to point protocol) is a dialup connection that must occur before a TCP/IP connection can be used. It is a feature that is used when ethernet is not available to dial in to a server for your internet access. This is not a feature that is used on a socket-by-socket basis. For example, calling Socket1.PPPConnect and Socket2.PPPDisconnect will affect all sockets on the system. In fact, saying Socket1.PPPDisconnect affects sockets that do not even belong to REALbasic! These features are system-wide features, and have been moved to the System object. Instead of using the Socket.PPP functions, please begin using their System counterparts. The functionality of these features remains the same.

Socket control can be instantiated via code since it is not a subclass of Control. This allows you to easily write code that does communications without adding the control to a window.

Writing to a socket is done asynchronously. This means each time the Write method is called, the data passed goes into a buffer in memory before actually being sent and then removed from the buffer. Once the socket has finished sending the data in the buffer to the computer at the other end of the socket connection, the SendComplete event handler is executed. This allows you to know when all of the data has really been sent.

Errors

Code	Message
100	There was an error opening and initializing the networking drivers.
102	Connection dropped.
103	The domain of the address was not found.
106	Invalid state error that is triggered when you attempt to use a socket in an incorrect fashion. An example would be trying to send data on a TCPSocket before the socket is connected or trying to join a multicast group with a UDPSocket before it is bound to a port.

If you receive an error message between -3211 and -3285, it is an OpenTransport error. Please refer to the error codes at http://developer.apple.com/qa/nw/nw35.html.

See Also

Object class.

SocketCore Class

The base class for TCPSocket, and UDPSocket. The SSLSocket class is derived from TCPSocket. **SocketCore** is an abstract class and cannot be instantiated directly. This group of classes supersedes the Socket control.

Super Class Object

Properties

Name	Type	Description
IsConnected	Boolean	Indicates whether the socket is currently connected. For TCPSockets, a connection means you can send and receive data, and are connected to a remote machine. For UDPSockets, this means that you are bound to the port and are able to send, receive, join or leave multicast groups, or set socket options.
LastErrorCode	Integer	The last error code for the socket.
LocalAddress	String	The local IP address of the computer.
Port	Integer	The port to bind on or connect to. On Mac OS X, attempting to bind to a port less than 1024 causes a SocketCore.Error event to fire with an error number 105 unless the application is running with root permissions. This is due to Apple's implementation with Open Transport, and currently, there are no workarounds.

Events

Name	Parameters	Description
DataAvailable		Occurs when additional data has come into the internal receive buffer.
Error		Occurs when an error occurs with the socket.
SendComplete	UserAborted as Boolean	Occurs when a send has completed. Use this to determine when all your data has been sent. *UserAborted* will always be False for UDP sockets.

Methods

Name	Parameters	Description
Close		Closes the socket's connection, closes any connections the socket may have, and resets the socket. The only information that is retained after calling Close is the socket's port, address (in the case of TCPSockets), and LastErrorCode properties. All other information is discarded.
Connect		Attempts to connect. For TCPSockets, the address and port properties must be set. For UDPSockets, the port property must be set. The Connect method binds a socket to a port.

Name	Parameters	Description
Poll		Polls the socket manually, which allows a socket to be used synchronously.
Purge		Removes all data from the socket's internal receive buffer. It does not affect the socket's internal send buffer.

Error Messages

The LastErrorCode Property returns one of the following error codes.

ErrorCode	Description
100	There was an error opening and initializing the drivers. Generally, this means that either Open Transport (on the Macintosh), or WinSock (on Windows) is not installed, or the version is too early.
101	This error code is no longer used. You will not see any 101 errors in version 5.0 and above.
102	This code means that you lost your connection. You will get this error if the remote side disconnects (whether its forcibly— by pulling their ethernet cable out of the computer), or gracefully (by calling SocketCore's Close method). This may or not be a true error situation. If the remote side closed the connection, then it is not truly an error; it's just a status indication. But if they pulled the ethernet cable out of the computer, then it really is an error; but the results are the same. The connection was lost. You will also get this error if you call the Disconnect method of TCPSocket.
103	REALbasic was unable to resolve the address that was specified. A prime example of this would be a mistyped IP address, or a domain name of an unknown or unreachable host.
104	This error code is no longer used. You will not see any 104 errors in version 5.0 and above.
105	The address is currently in use. This error will occur if you attempt to bind to a port that you have already bound to. An example of this would be setting up two listening sockets to try to listen on the same port.
106	This is an invalid state error, which means that the socket is not in the proper state to be doing a certain operation. An example of this is calling the Write method before the socket is actually connected.
107	This error means that the port you specified is invalid. This could mean that you entered a port number less than 0, or greater than 65,535. It could also mean that you do not have enough privileges to bind to that port. This happens under Mac OS X if you are not running as root and try to bind to a port below 1024. You can only bind to ports less than 1024 if you have Root privileges in OS X. A normal "Admin" user does not have Root privileges.

These are not the only errors that are returned by LastErrorCode. If REALbasic cannot adequately map the provider's error code to one of the above codes, it will pass you the provider's error code. For the Macintosh, these error codes are negative numbers in the range [-3211, -3285]. For Windows, these error codes are usually positive numbers, in

the range [10004, 11004]. For a description of the Macintosh error codes, please see MacErrors.h. For Windows error codes, see WinSock.h.

Notes The SendComplete event's parameter lets you know whether the transfer has completed or has been cancelled by returning True from the SendProgress event of the TCPSocket class. You can use this information to update different status variables, or alert the user of transfer success or failure. If the user aborted, this parameter is True, and if the send was completed, this value is False. This is a new feature to both Macintosh and Windows. The SendComplete event's parameter is always False for UDPSockets since there is not a SendProgress event for that class.

See Also TCPSocket, UDPSocket, DataGram classes.

Sort Method

Sorts the elements of an array in ascending order.

Syntax ***array*.Sort**

Part	Description
array	Required. The array to be sorted.

Notes The **Sort** method works with Integer, string, single, and double arrays only.

Examples This example sorts the aNames array.

```
aNames.Sort
```

See Also Dim statement, Insert method, Redim method, Remove method, UBound function.

Sound Class

Used to play sounds.

Super Class

Object

Methods

Name	Parameters	Description
IsPlaying		Returns True if the sound is playing.
Pan		Specifies the relative volume (balance) between the left and right speakers. Range is from -100 to +100, with 0 indicating equal volume in the left and right channels. -100 plays sound in the left channel only; 100 plays sound in the right channel only. The default is 0.
Play		Plays the sound.
PlayLooping		Plays the sound in an infinite loop.
Stop		Stops a sound that is playing.
Volume		Controls the volume of the sound. Range is from 0 to 100; 0 mutes the sound and 100 plays the sound at the "normal" volume set in the computer's volume setting. Default is 100.

Notes

Sounds that have been dragged into the Project window can be accessed via their object name. Sounds can also be accessed by calling the OpenAsSound method of a FolderItem. There are no properties or events for sound objects.

The Macintosh version supports .wav files as well as Macintosh .snd resources.

Examples

This example plays a sound called "Sledgehammer" which has been dragged into the Project window.

```
sledgehammer.play
```

This example loads a sound file called "Doh!" from the current directory (folder) into a sound object and plays it.

```
Dim f as FolderItem
Dim s as Sound
f = GetFolderItem("Doh!")
s = f.OpenAsSound
s.play
```

The following example plays the sound "Giggle", which has been dragged into the Project Window, in an endless loop.

```
Giggle.playlooping
```

This example stops "Giggle."

```
Giggle.stop
```

See Also

NotePlayer control.

Sprite Class

A picture that can be animated by a SpriteSurface control.

Super Class

Object

Properties

Name	Type	Description
Group	Integer	Determines whether or not the SpriteSurface control's Collision event handler will be called when the Sprite comes in to contact with another Sprite. Sprites with a negative Group value will collide with any other Sprites (except those with a Group value of 0). A Group value of 0 means the Sprite will not collide with any other Sprites. A positive Group value means that the Sprite will collide with Sprites of any other Group value (except 0) but not those with the same Group value.
Image	Picture	The picture that will be drawn when the SpriteSurface's Run method is called.
Priority	Integer	Determines the order the Sprite is drawn in during the NextFrame event handler.
ScreenX	Integer	The sprite's distance in pixels from the left side of the SpriteSurface.
ScreenY	Integer	The sprite's distance in pixels from the top of the SpriteSurface.
X	Integer	The location of the left side of the Image within the SpriteSurface area.
Y	Integer	The location of the top of the Image within the SpriteSurface area.

Methods

Name	Parameters	Description
Close		Removes the Sprite from the SpriteSurface it was created in.

Notes

A **Sprite** object is a picture that can be animated using a SpriteSurface control. Moving the Sprite across the screen is simply a matter of changing the Sprite's x and y

properties in the NextFrame event handler of the SpriteSurface control that created the sprite. Should two sprites touch each other during the animation, there is the potential for a *collision*. A collision means that the SpriteSurface's Collision event handler will be executed and the two Sprites that have collided will be passed to it. This gives you the opportunity to decide what should happen when the two sprites come into contact. Whether or not a collision occurs is based on the values of the two Sprites' Group properties:

Group Values	Collision?
0 for either Sprite	No
Both Sprites have same positive value	No
Both Sprites have different positive values	Yes
One Sprite has a negative value	Yes

Sprites are drawn during the SpriteSurface NextFrame event handler's execution in order based on their Priority property. This property is assigned by default based on the order the Sprites were created. The first Sprite's Priority is 1, the second Sprite's Priority is 2, etc. In this example, when the first and second Sprites pass over each other, the second Sprite will appear to be in front of the first Sprite.

All Sprite object properties are changeable at runtime. This means you can change the Sprite's image, location, group, and priority during the animation.

A Sprite can be created using the New operator and attached to a SpriteSurface using the Attach method of the SpriteSurface class.

Comparing ScreenX and X (and ScreenY and Y)

ScreenX and X (and ScreenY and Y) will be the same for a SpriteSurface that has not been scrolled. When it is scrolled, the sprite's ScreenX and ScreenY values change because their positions on the screen have changed. Their X and Y values remain the same because their position in "world" coordinates has not changed.

Mask and Transparency Properties

Sprites ignore the Mask property of the Picture, as well as the value of the Transparent property. White pixels are always transparent and non-white pixels are always opaque.

Examples

See the SpriteSurface control's example.

See Also

SpriteSurface control.

SpriteSurface Control 

Used to create sprite animation.

Super Class RectControl

Properties

Name	Type	Description
BackDrop	Picture	A picture that will be drawn within the area defined by the Height, Left, Top, and Width properties when the SpriteSurface Run method is called. If the picture is too small, it will be tiled. to fill out the entire area. Sprite objects will be drawn in front of the BackDrop.
ClickToStop	Boolean	If True, clicking the mouse will stop the SpriteSurface animation.
Depth	Integer	Bit depth of the spritesurface area. May take on values of 0, 8,16, or 32. Specifying 0 causes the spritesurface to automatically match its bit depth to the depth of the monitor at the time the spritesurface was opened. The Depth property can be changed on the fly.
FrameSpeed	Integer	The number of vertical retraces per frame. Zero is the fastest the computer can redraw. Compute FrameSpeed by dividing 60 by the number of frames per second you want and round to the next Integer. Each frame will cause the NextFrame event to execute. The definition of FrameSpeed has changed in version 2 of REALbasic. Please update version 1 applications accordingly.
Graphics	Graphics	Used for drawing to the screen when the SpriteSurface is running. Avoid drawing into the area defined by the *Height*, *Left*, *Top*, and *Width* properties since the SpriteSurface handles this area.
Height	Integer	The height of the spritesurface control in local coordinates (relative to the window).
Index		The number of the control when it's part of a control array.
Left	Integer	The left side of the spritesurface control in local coordinates (relative to the window).
LockBottom	Boolean	Determines whether the bottom edge of the spritesurface should stay at a set distance from the bottom edge of the owning window.
LockLeft	Boolean	Determines whether the left edge of the spritesurface should stay at a set distance from the left edge of the owning window. *LockLeft* has no effect unless *LockRight* is True.
LockRight	Boolean	Determines whether the right edge of the spritesurface should stay at a set distance from the right edge of the owning window.

Name	Type	Description
LockTop	Boolean	Determines whether the top edge of the spritesurface should stay at a set distance from the top edge of the owning window. *LockTop* has no effect unless *LockBottom* is True.
Name	String	The name of the control.
ScrollX	Integer	Distance the SpriteSurface has been scrolled in the horizontal direction.
ScrollY	Integer	Distance the SpriteSurface has been scrolled in the vertical direction.
Super		The class of object the control is based on.
SurfaceHeight	Integer	Now set equal to *Height*.
SurfaceLeft	Integer	Now set equal to *Left*.
SurfaceTop	Integer	Now set equal to *Top*.
SurfaceWidth	Integer	Now set equal to *Width*.
Top		The top of the spritesurface control in local coordinates (relative to the window).
Width		The width of the spritesurface control in local coordinates (relative to the window).

Events

Name	Parameters	Description
Collision	S1 as Sprite, S2 as Sprite	Two sprites with different Group property values (or negative Group property values) have collided (one has touched the other).
NextFrame		The SpriteSurface is ready to draw the next frame of animation. Sprite objects will be redrawn at the end of this event handler.
PaintTile	g as Graphics xpos as Integer ypos as Integer	Occurs when the Scroll method is called. g is a Graphics object and *xpos* and *ypos* are the coordinates of the square that needs to be redrawn.

Methods

Name	Parameters	Description
Attach	S as Sprite	Allows you to attach a Sprite to a SpriteSurface.
Close		Stops the animation and returns the screen to the window that contains the SpriteSurface control.
KeyTest	KeyCode as Integer	Returns True if the key represented by the *KeyCode* passed has been pressed. See the KeyCode chart for acceptable values.
NewSprite	Image as Picture, X as Integer, Y as Integer)	Returns a new Sprite object and places its Image within the SpriteSurface at the coordinates passed.

Name	Parameters	Description
PaintTile	X as Integer Y as Integer	Forces the background tile specified by x,y to be repainted, triggering your PaintTile event. If you need to change part of the background during a game, this is the best way to force the background to update. (Note that the update will not actually appear on the screen until the next frame.)
Run		Begins the animation.
Scroll	dx as Integer dy as Integer	Scroll the SpriteSurface the amount given by dx and dy.
Stop		Stops the animation.
Update		Runs the animation one step per call. It redraws the sprites that have moved or changed, fires collision events, and calls the NextFrame event once.

Notes

Updating REALbasic 2.1 (and earlier) Projects

When you open a project that uses a SpriteSurface control created with an earlier version of REALbasic, any SpriteSurface control will have a height and width of zero. This makes it difficult to select using the mouse. Use Edit ▶ Select All (Command-A) to select it and then use the Properties window to resize it.

A Selected SpriteSurface object brought over from v2.1

To resize the SpriteSurface control so that it 'takes over' the user's monitor completely, use the FullScreen and MenuBarVisible properties of the parent window, as described in the following section.

Please note that the SurfaceWidth, SurfaceHeight, SurfaceLeft, and SurfaceTop properties are obsolescent.

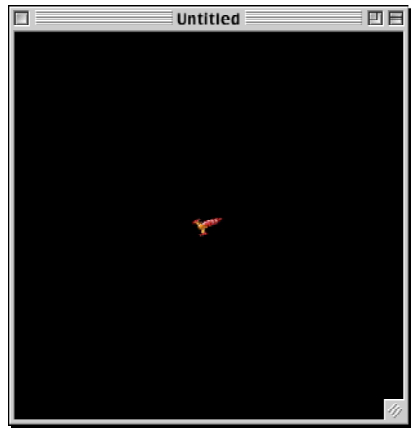
In previous versions of REALbasic, you used the Run method to begin the animation. It ran until it was closed and then it disappeared. In version 3.0, you can use the Update method to run the animation one step at a time. Rather than using Run, you can call Update repeatedly, perhaps from a Timer control. Also, the ClickToClose property has become CickToStop.

Sizing and Positioning the SpriteSurface Area

The SpriteSurface control is now a "regular" visible control that can be dragged from the Controls palette onto a window and resized by dragging or by assigning values to the The Left, Top, Width, and Height properties. These properties are read/write and can be used to change the size and location of the SpriteSurface object on-the-fly.

You can also use two properties of the parent window, FullScreen and MenuBarVisible, to allow the SpriteSurface object to take over the user's entire screen area. Set FullScreen to True and MenuBarVisible to False and use the SpriteSurface's Lock… properties to lock the SpriteSurface control to the parent window so that the SpriteSurface area expands automatically. This sets the Left, Top, Width, and Height properties accordingly.

The following illustration shows a SpriteSurface configured so that it expands to fill the entire screen when FullScreen is True.



REALbasic Version 1 compatibility

The definition of FrameSpeed has changed from version 1 to version 2. If you are converting a version 1 application to version 2, you need to check your code for usages of FrameSpeed and recalculate its value given the new definition.

Creating Animation with Sprites

The SpriteSurface control is used to create animation where pictures can be moved around the screen with all redrawing handled automatically by the SpriteSurface control. Each picture is a Sprite object. Sprite objects have x and y properties that determine their current location on the screen when the SpriteSurface is running the sprite animation.

Causing Sprites to Move and Change Images

The NextFrame event handler is called each time the SpriteSurface is ready to draw the next frame of animation. If you want a Sprite to change position in the next frame, change its x and/or y properties in the NextFrame event handler. If you want the Sprite's picture to change in the next frame of animation, change its Image property in

the NextFrame event handler. You can use the Update method to run the animation one step at a time. It calls the NextFrame event handler only once.

To remove a Sprite from the animation, simply call the Sprite's Close method.

Frame Redrawing

The speed at which frames are redrawn is based on the FrameSpeed property. The higher the FrameSpeed, the faster the animation will run.

To begin the animation, call the SpriteSurface's Run method. To stop the animation, call the SpriteSurface's Close method.

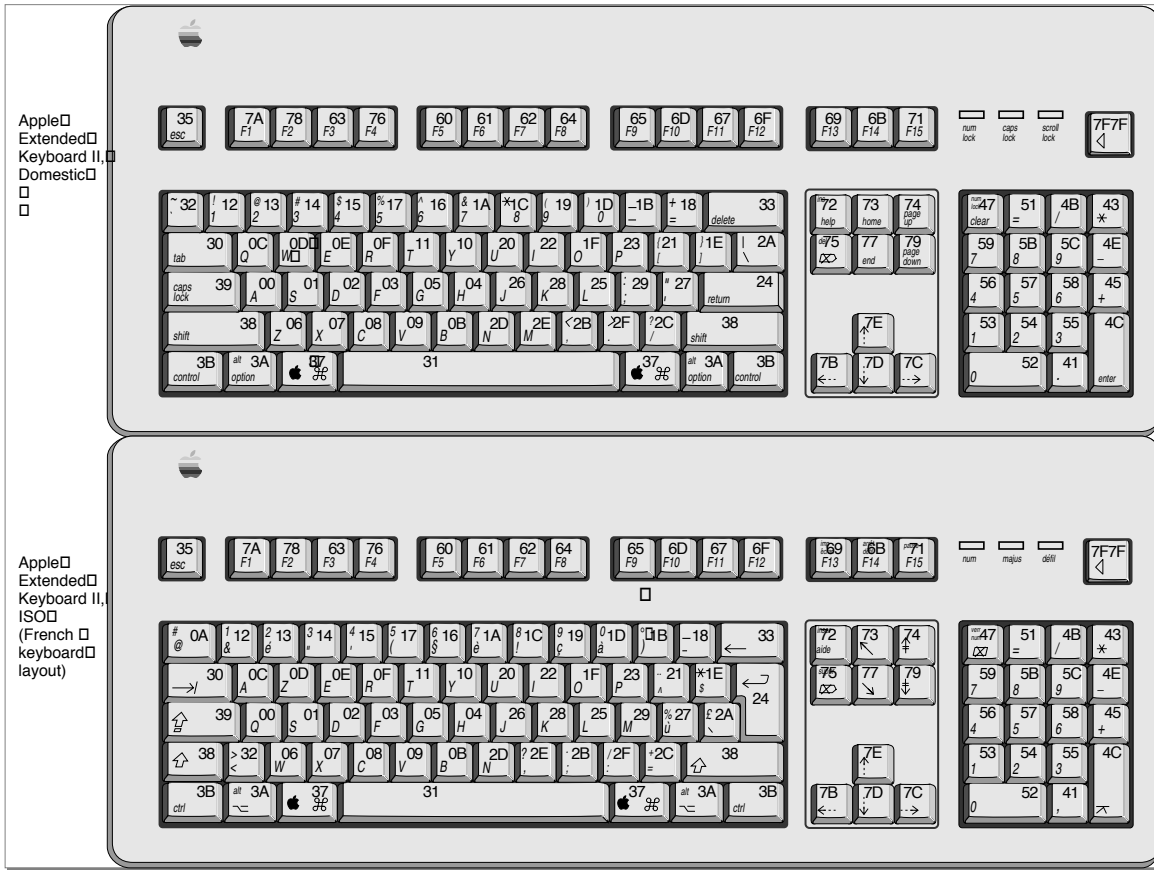
Current Implementation

The size and position of the SpriteSurface area are controlled by the Left, Top, Height, and Width properties or by setting the parent window's FullScreen property to True and using the LockLeft, LockTop, LockRight, and LockBottom properties to automatically expand the SpriteSurface area as the window expands.

Once the Run method is called, NextFrame events will continue to be called until the user clicks the mouse button if the ClickToStop property is True. If ClickToStop is False, the animation will continue until the window that the SpriteSurface is on closes.

KeyCodes

The following graphic shows the keycodes for an Apple standard keyboard and the French keyboard. This graphic was provided courtesy of Apple Computer, Inc.:



Examples

In this example, a Sprite is moved back and forth horizontally across the screen. The PushButton's Action event handler creates a Sprite and stores it in a window property of type Sprite named LisaSprite. The image for the LisaSprite object is a pict image called LisaPict that has been dragged into the Project window. The SpriteSurface control named SpriteSurface1 will control the animation. The PushButton then begins the animation by calling the Run method of the SpriteSurface1 control. Another window property of type Integer named LisaDirection is used to keep track of the direction of the LisaSprite. Here are the event handlers:

Pushbutton1:

```
Sub Action()
 LisaSprite=SpriteSurface1.NewSprite(LisaPict,200,200)
  LisaDirection=0
  SpriteSurface1.Run
End Sub
```

SpriteSurface1:

```
Sub NextFrame()
  If LisaSprite.x>=640 Then
    LisaDirection=1
  ElseIf lisaSprite.x<=0 then
    lisaDirection=0
  End if
  Select Case lisaDirection
  Case 0
    lisaSprite.x=LisaSprite.x+1
  Case 1
    lisaSprite.x=lisaSprite.x-1
  End Select
End Sub
```

This example closes (using the Close method) the running SpriteSurface control when the user presses the Return key:

```
Sub NextFrame()
  If Me.KeyTest(&h24) Then
    Me.Close
  End if
End Sub
```

A more sophisticated example called “REALSprites” that uses a new class to control sprites is available at our ftp site and on the REALbasic CD.

The following example uses the PaintTile event to animate the screen image when the user presses an arrow key. The PaintTile event occurs when the scroll method is called.

The SpriteSurface's NextFrame event has the following code:

```
If spriteSurface1.keytest(&h7E) then //up
 me.scroll 0 , -10
elseif spriteSurface1.keytest(&h7D) then //down
 me.scroll 0 , 10
elseif spriteSurface1.keytest(&h7B) then //left
 me.scroll -10,0
elseif spriteSurface1.keytest(&h7C) then //right
 me.scroll 10,0
elseif spriteSurface1.keytest(&h24) then //quit
 me.close
end if
```

Each condition tests whether the user pressed an arrow key and calls the scroll method with appropriate x and y values. The PaintTile event has the following code:

```
Sub PaintTile (g as Graphics, xpos as Integer, ypos as Integer)
 g.foreColor=rgb(0,0,0) //set the forecolor to black
 g.fillrect 0,0,64,64 //erase the tile
//set the forecolor to white
 g.foreColor=rgb(255,255,255)
 g.drawstring str(xpos)+"/"+str(ypos), 0,32
end sub
```

See Also

Sprite object.

Sqrt Function

Returns the square root of the value specified.

Syntax

***result*=Sqrt (*value*)**

Part	Type	Description
value	Double	The value you want the square root of.
result	Double	The square root of the *value*.

Notes

The **Sqrt** function returns the square root of the value passed to it.

Examples

This example uses the **Sqrt** function to return the square root of a number.

```
Dim d As Double
d=Sqrt(16) //returns 4
```

SSLSocket Control

Use the **SSLSocket** control to do secure communications via TCP/IP using secure sockets layer (SSL) technology. The **SSLSocket** control is available only in the Professional version of REALbasic. The Standard version of REALbasic does not compile references to the **SSLSocket** control.

SSLSocket currently does not support listening secure sockets.

Super Class

TCPSocket class.

Properties

Name	Type	Description
ConnectionType	Integer	Specifies the type of SSL connection. ConnectionType can take the following values: 0 - SSLv2: SSL (Secure Sockets Layer) version 2. 1 - SSLv23: SSL version 3, but can roll back to 2 if needed. 2- SSLv3: SSL version 3. 3- TLSv1: TLS (Transport Layer Security) version 1. The default value is 2 (connection type SSLv23). Once you have called the SSLSocket.Connect method, you cannot change the connection type without calling SSLSocket.Close first, but you can always query the connection type.
Secure	Boolean	Set to True to specify an SSL connection. If Secure is False, the SSLSocket transmits data just like a TCPSocket.
SSLConnected	Boolean	True if you have an SSL connection.
SSLConnecting	Boolean	True if REALbasic is in the process of doing a handshake to establish an SSL connection.

Events

Name	Parameters	Description
Connected		A connection has been established using the Address and Port properties.
DataAvailable		New incoming data is now available in the buffer.
Error		An error occured.
SendComplete		All of the data in the socket buffer has been written (sent).

Notes

To establish an SSL connection, set the Secure property to True and use the Connect method.

Currently, setting the Secure property to True and then calling the Listen method will not create a secure listening socket. Doing so will create a non-secure listening port on

your machine at the port specified, and any connections received will not be secure. A secure listening feature may be added later to REALbasic.

The **SSLSocket** control does not have an icon of its own in the Controls palette. There are two ways to add an **SSLSocket** control to a window:

- Drag a TCPSocket control to a window and then change its Super class to **SSLSocket**.
- Display the window's contextual menu by control-clicking on the window (Macintosh) or right-clicking (Windows) and then choose Add ▶ SocketCore ▶ TCPSocket ▶ SSLSocket.

If you drag from the Controls palette, you will get the Socket control's icon. This difference is only cosmetic and does not affect the functionality of the SSLSocket.

The **SSLSocket** control can be instantiated via code since it is not a subclass of Control. This allows you to easily write code that does communications without adding the control to a window.

Writing to a socket is done asynchronously. This means each time the Write method is called, the data passed goes into a buffer in memory before actually being sent and then removed from the buffer. Once the socket has finished sending the data in the buffer to the computer at the other end of the socket connection, the SendComplete event handler is executed. This allows you to know when all of the data has really been sent.

Calling Read, ReadAll, or Lookahead may not fetch all of the data in the internal buffer. This is because SSL needs to read data in blocks (due to the cryptography), and it may not have a complete block in the buffer. For example, there may be 700 bytes available in the buffer, but SSL can only decrypt 512 bytes due to the remainder being an incomplete block. What occurs in this case is some data may remain stagnant in the buffer. When more data comes in, REALbasic executes a DataAvailable event. If there are no more DataAvailable events, then upon disconnection, REALbasic will execute additional DataAvailable events to let you pick up any stagnant data that SSL can give us back. There are two things to watch out for because of this:

1) If there is not sufficient data for SSL to decrypt, you may get a DataAvailable event but no data.
2) Calling SSLSocket.Close may execute DataAvailable events.

See Also

SocketCore, TCPSocket, Object classes.

StackOverFlowException Error

Occurs when the calling chain becomes too long.

Super Class

RuntimeException

Notes

Not surprisingly, a **StackOverflowException** occurs when the stack overflows. This happens when the calling chain gets too long. This can easily happen when your code makes a recursive call without providing a way to terminate the recursion—or the condition that terminates the recursive call takes too many calls to occur.

Example

The following method calls itself until the stack overflows:

```
Function Square (value as Integer) as Integer
 Return Square(value)
End Function
```

You can handle the function with the following simple Exception handler:

```
Function Square (value as Integer) as Integer
 Return Square(value)
Exception err
 If err IsA StackOverflowException then
  MsgBox "The stack has overflowed!"
 end if
End Function
```

See Also

RuntimeException class, OutofBoundsException error, IllegalCastException error, TypeMismatchException error, NilObjectException error, Function statement, Raise statement, Nil function, Exception block.

StandardToolbarItem Control

Used to create a toolbar in a window. Supported on Mac OS X 10.2 and above only. StandardToolbarItems can be placed in windows for other platforms for layout purposes only.

Super Class

Control

Properties

Name	Type	Description
ControlOrder	Integer	Determines the left to right placement of the item in the toolbar.
Enabled	Boolean	True if the item is enabled.
Handle	Integer	A ToolbarItemRef, used in Declare statements, of interest only to toolbox programmers.
IdentifierOther	String	A toolbarItem identifier that REALbasic hasn't provided. (e.g. another way to make a separator is to set this property to "com.apple.hitoolbox.toolbar.separator").
Left	Integer	The distance (pixels) from the left side of the window in the Window Editor. In the runtime application, the ordering of toolbar items is strictly based left to right on control order.
Top	Integer	The distance (pixels) from the top of the window in the Window Editor. In the runtime application, the ordering of toolbar items is strictly based left to right on control order.
Type	Integer	Type - One of the Apple pre-defined toolbar types or a custom toolbar type: 0 - Other (you need to use IdentifierOther to specify the identifier to use) 1 - Separator (cosmetic widget only) 2- Space, separates items by a small space 3 - Flexible Space, pushes items after it to be right justified

Events

Name	Parameters	Description
Close		Occurs when the item is destroyed
Open		Occurs when the item is first created.

Note

If at least one ToolbarItem or **StandardToolbarItem** is on the window, a toolbar will be created, and the toolbar button will be added to the window. Each ToolbarItem will be added in Control Order from Left to Right.

See Also

ToolbarItem control.

StartupItemsFolder Function

Used to access the StartupItems folder in the System folder on Macintosh.

Syntax

***result*=StartupItemsFolder**

Part	Type	Description
result	FolderItem	A FolderItem that represents the Startup Items Folder.

Notes

Use the **StartupItemsFolder** function to access the Startup Items folder in the user's System Folder.

Items in the Startup Items folder are opened automatically when the user starts the Macintosh. If want your application to give the user the option to automatically open it at system startup, you will need to place it (or an alias to it) in the Startup Items folder.

The **StartupItemsFolder** function provides a way to access the Startup Items folder that will work under different language systems, and will continue to work even if Apple reorganizes the System folder.

If your application will run on Windows, you can first test the global constant TargetWin32 to determine whether the application is on a Windows machine.

Examples

See the AppleMenuFolder function for an example.

See Also

AppleMenuFolder, ApplicationSupportFolder, ControlPanelsFolder, DesktopFolder, ExtensionsFolder, FontsFolder, PreferencesFolder, SystemFolder, TemporaryFolder, TrashFolder functions.

StaticText Control

Displays text that is not editable by the user.

Super Class

RectControl

Because this is a RectControl, see the RectControl for other properties and events that are common to all RectControl objects.

Properties

Name	Type	Description
Bold	Boolean	Applies the bold style to the text.

Name	Type	Description
Caption	String	The text displayed. Same as the Text property. This property was added for Visual Basic compatibility.
DataField	String	Relevant only if the StaticText is used in conjunction with a DataControl to display the contents of fields in a database table. Name of a field in the table referenced by DataControl.
DataSource	DataControl	The DataControl that references a table in a database whose fields are displayed (TableName as String). If assigned in code, you use the Name property of a DataControl. Use the DataField property to link a field to be displayed to a StaticText. In the IDE, a popup menu of field names will appear. Database fields can also be displayed using EditFields, PopupMenus and ListBoxes.
Italic	Boolean	Applies the italic style to the text.
Multiline	Boolean	If True, wrap text to multiple lines.
Text	String	The text displayed. Same as the Caption property.
TextAlign	Integer	The alignment of the text: 0 = left 1 = middle 2 = right
TextColor	Color	The color of the text.
TextFont	String	Name of the font used to display the text.
TextSize	Integer	Size of the font used to display the text.
Underline	Boolean	Applies the underline style to the text.

Events

Event	Parameters	Description
AcceleratorKey		The accelerator key for the **StaticText** was pressed.
MouseDown	x as Integer, y as Integer	The mouse button was pressed inside the **StaticText** region at the location passed in x and y. Return True if you are going to handle the MouseDown.
MouseDrag	x as Integer, y as Integer	The mouse button was pressed inside the **StaticText** and moved (dragged) at the location local to the control passed to x and y. This event will not occur unless you return True in the MouseDown event.
MouseUp	x as Integer, y as Integer	The mouse button was released inside the **StaticText** region at the location passed in to x,y. This event will not occur unless you return True in the MouseDown event.

Notes

On Windows, you can define an Accelerator key for the control. In the Caption property, precede the accelerator with an ampersand. For example, the caption "&Home" signifies that the H is the accelerator. Pressing Alt+H on Windows triggers the Action event.

See Also

RectControl class; EditField control.

Str Function

Returns the string form of the value passed.

Syntax

***result*=Str(*number*)**

Part	Type	Description
result	String	The string version of the number passed.
number	Integer, Single, or Double	Any numeric expression.

Notes

Use the Format function when you want to convert numeric values into formatted strings such as dates, times, currency etc.

Str returns seven (7) significant digits. It begins using scientific notation when it needs to return more than 6 digits.**Str**(123456789) returns 1.234568e+8.

Str assumes the period (.) is the decimal separator. If your application needs to recognize other decimal separators, use the CStr function.

Examples

This example uses the **Str** function to return the string form of several numbers.

```
Dim s As String
s=Str(123)      //returns "123"
s=Str(-123.44)  //returns "-123.44"
s=Str(123.0045) //returns "123.0045"
Const Pi=3.14159265358979323846264338327950
s= Str(pi)  // returns "3.141593"
```

See Also

CDbl, CStr, Val, Format functions.

StrComp Function

Makes a binary (case-sensitive) or text (lexicographic) comparison of the two strings passed and returns the result.

Syntax

***result*=StrComp(*string1*, *string2*, *mode*)**

OR

***result*=*stringVariable*.StrComp(*string2*, *mode*)**

Part	Type	Description
result	Integer	If *string1* < *string2* the function returns -1 If *string1* = *string2* the function returns 0 If *string1* > *string2* the function returns 1
string1	String	The first string for comparison.
string2	String	The second string for comparison.
mode	Integer	The comparison mode. 0=binary (case-sensitive) and 1=text (lexicographic)
stringVariable	String	Any variable of type String.

Notes

The text comparison is lexicographic, not case-insensitive. The means that case will be taken into account if the strings are the same (other than case).

When using Mode 0 (binary), **StrComp** always compares the bytes of the strings as they are, doing no encoding conversions. It's possible that two strings might look the same on screen but be reported as different in **StrComp** Mode 0 if they are in different encodings (and, therefore, contain different bytes).

Examples

The following example returns -1 because the two strings are the same in every way except in case.

```
StrComp("Spam", "spam", 1)
```

The following example returns -1 because in a text comparison of the two strings, *string1* is greater than *string2*. The ASCII value of "s" is greater than the ASCII value of "S".

```
StrComp("Spam", "spam", 0)
```

See Also

InStr function; >, >=, <, <=, =, <> operators.

String Data Type

A **String** is an intrinsic data type in REALbasic. It is a series of numeric or alphabetic characters enclosed in quotes. Any kind of alphabetic or numeric information can be stored as a **string**. “Jean Marie”, “3/17/98”, “45.90” are all examples of strings. When you place quotes around information in your code, you are telling REALbasic to look at the data as just a series of characters and nothing more. The maximum length of a **string** is limited only by available memory.

REALbasic includes an extensive library of functions for comparing and manipulating strings, which are listed in the See Also section.

The VarType function returns 8 when passed a **String**.

Example

```
Dim s as String
s="How now brown cow"
```

See Also

+, <, <=, <>, =, >, >=, Asc, AscB, Chr, ChrB, CStr, InStr, InStrB, Left, LeftB, Len, LenB, Lowercase, LTrim, Mid, MidB, NthField, Replace, ReplaceAll, Right, RightB, RTrim, Str, StrComp, Titlecase, Trim, Uppercase, Val, VarType functions.

StringProvider Interface Class

Used to program custom object bindings.

Methods

Name	Type	Description
getString	String	Obtains current value of string.
setString	String	Sets value of string.

StringShape Class

Draws a text string in a vector graphics environment.

Super Class

Object2D

Properties

Name	Type	Description
Bold	Boolean	Set to True to use the Bold style.
Italic	Boolean	Set to True to use the Italic style.
Text	String	The text to draw. The text can be only one line and in one text style (font, font size, style) specified by the other StringShape properties.
TextFont	String	Name of the font to use.
TextSize	Double	Font Size in points.
Underline	Boolean	Set to True to use the Underline style.

Notes

The X,Y properties specify the center of the string's baseline. Strings that contain line breaks are not supported. **StringShapes** can be rotated, but doing so is memory intensive, especially for large strings.

Example

This example draws a line of text when the user presses the mouse button. Place the following code in the MouseDown event of a window.

```
Dim s as StringShape
s=New StringShape
s.Text="Hello World"
s.TextFont="Helvetica"
s.Bold=True
Graphics.DrawObject s,x,y
```

See Also

ArcShape, CurveShape, FigureShape, FolderItem, Group2D, Graphics, Object2D, OvalShape, Picture, PixmapShape, RectShape, RoundRectShape classes.

StyledTextPrinter Class

Used to print styled text stored in an EditField.

Super Class

Object

Properties

Property	Type	Description
EOF	Boolean	True when there is no more text to print in the EditField. Use it to determine when to stop calling DrawBlock. See examples.
Width	Integer	Width of text to be printed.

Methods

Method	Parameters	Description
DrawBlock	x as Integer y as Integer Height as Integer	Used to print the styled text. *x* and *y* are coordinates of the point at which you want to begin printing the text. *x* and *y* are offset from the top-left corner of the printable area as defined in Page Setup.

Examples

The following example prints the styled text in an EditField. The EditField's MultiLine property must be set to True to accept and print styled text.

```
Dim stp as StyledTextPrinter
Dim g as Graphics
g=OpenPrinterDialog()
If g <> Nil then
 stp=EditField1.StyledTextPrinter(g,72*7.5)
 stp.DrawBlock 0,0,72*10
End if
```

This example prints the contents of EditField1 in two columns with a quarter-inch of spacing between them. The EOF property is used to determine if all the text has been printed. When EOF is False, it prints the second column or prints another page.

```
Dim g as Graphics
Dim stp as StyledTextPrinter
Dim ColumnWidth, SpaceBetweenColumns, PageHeight as  Integer
Dim ColumnToPrint as Integer


ColumnWidth = 261
//7.5 inches minus 1/4 inch for space
//divided by 2
SpaceBetweenColumns = 18
//18 pixels is 1/4 inch
PageHeight = 10 * 72
//10 inches * 72 pixels per inch

g=OpenPrinterDialog()
If g <> Nil then
 stp=EditField1.StyledTextPrinter(g,540)
 stp.width = ColumnWidth

 ColumnToPrint = 1
 Do until stp.eof
  stp.DrawBlock (columnWidth+SpaceBetweenColumns)*
    (columnToPrint-1),0,PageHeight
  If ColumnToPrint = 2 then //printing last column
   If Not stp.eof then // more text to print
    g.nextPage
    ColumnToPrint = 1
   End if
  Else // more columns to print on this page
   ColumnToPrint = columnToPrint + 1
  End if
 Loop
End if
```

See Also

EditField control; OpenPrinter, OpenPrinterDialog functions; Graphics, PrinterSetup classes.

Sub Statement

Declares the name, parameters, and code that form the body of a subroutine (method).

Syntax

Sub *name*([*parameterList*])

[local variable declarations]

[statements]

[Return]

[statements]

[*exception handlers*]

[Finally]

End Sub

Part	Description
name	Required. The name of the subroutine (method); follows standard variable naming conventions.
parameterList	Optional. List of values representing parameters that are passed to the Subroutine when it's called. Multiple parameters are separated by commas.

Notes

The **Sub** statement is used to define a method. The word **Sub** is used because in the BASIC language, a subroutine is the equivalent of a method. Methods are usually associated with an object (Exceptions, for example, are global methods or functions that are part of a module). All executable code must be in a **Sub** or Function statement. A **Sub** can not be defined inside another **Sub** or Function. A **Sub** executes each line of code from the top down. Once the last line of code is executed, control returns to the line following the statement that called the **Sub**.

You call a **Sub** by using its name followed by any parameters in parentheses.

All variables used in a **Sub** must be declared before they are used in a statement. Variables used in a **Sub** are either global or local. Global variables can be accessed from within any **Sub** or Function. They exist from the moment the application runs to the time it quits. Global variables are created by calling the Dim statement within the **Sub** of a module. Local variables are variables that are created each time the **Sub** is run and destroyed when the **Sub** finishes. Consequently, they can only be accessed by the statements within the **Sub**. They are created by using the Dim statement from within a **Sub** or Function except those attached to a module (as that would make them global variables).

The **Return** statement can be used to immediately return control to the statement that called the **Sub**.

Exception handlers are statements that handle errors. See the RuntimeException class and the Exception Block for more information.

Sometimes a method needs to do some cleanup work whether it is finishing normally or aborting early because of an exception. The optional "Finally" block at the end of the method runs after the exception handlers, if it has any. Code in this block will be executed even if an exception has occured, whether the exception was handled or not.

If you need your method to return one value, you will want to declare it as a Function. A Function is a method that can return a value. See the Function statement for more information.

When your method must return more than one value, you must use another approach. Parameters can be passed by value or by reference. By default, parameter passing is done by value and the value cannot be modified by the method. When you pass a parameter by reference, a pointer to the object is passed. This allows you to return a value using the parameter.

If you want to pass a parameter by reference, precede it by the keyword 'ByRef' in the parameter list, e.g., 'ByRef MyInt as Integer').

A **Sub** method can call itself resulting in recursion. Too much recursion can lead to StackOverFlowException errors.

See Also Exit statement, Function statement, RuntimeException class, Exception Block.

Super Cannot be used in a Module Method Error

The concept of Super applies only to the Class hierarchy, so it has no meaning in a method of a module.

Syntax Error Error

Any number of miscellaneous errors that were not identified by more specific error messages has occurred.

Examples An omitted closing parenthesis:

```
If (d > 0 then
.
end if
```

Used a Dim statement inside a conditional statement:

```
Dim s as String
s="A"
if s="A" then
 Dim b as Boolean
 b=False
end if
```

An incorrect Dim statement:

```
Dim d Double //omitting 'as'
Dim f //omitting 'as' and data typ
Dim As Boolean //no variable name
Dim Double as Double //'Double' is a reserved word
Dim a b as Integer //should be 'a,b'
Dim c as //type missing
Dim myFlag as True //illegal type
```

The 'Then' keyword is missing from an If statement:

```
Dim a as Integer
a=5
If a >0 //needs to be If a>0 then
 MsgBox "Boo"
end if
```

The Elseif statement was not preceded by an If statement.

```
Dim i,j,k as Integer
// do something
elseif j>0 then
//do something else
end if

//Correct
If j=0 then
.
Elseif j>0 then
.
End if
```

The End If keyword is missing from an If statement:

```
Dim dayNum as Integer
if dayNum=1 then
 MsgBox "It's Monday"
// end if needed
```

#else used instead of Else with a #If statement (conditional compilation).

```
Dim c as Color
#If TargetMacOS then
 b=SelectColor( c ,"Select a Color")
else // should be #else
 Beep
#EndIf
```

The Next keyword is missing from a For loop:

```
Dim i,j as Integer
Dim aInts(2,2) as Integer
For i=0 to 2
 For j=0 to 2
  aInts(i,j)=i*j
 Next
//final 'Next' missing here
```

One too many Next keywords are used in this example:

```
Dim i,j as Integer
Dim aInts(2,2) As Integer
For i=0 to 2
 For j=0 to 2
  aInts(i,j)=i*j
 Next
Next
Next //too many "nexts"
```

Omitting the Wend keyword from a While loop:

```
While i<10
 Beep
Next //should be Wend
```

Omitting the While statement:

```
Dim i as Integer
i=1
//While statement missing here
  Beep
  i=i+1
Wend
```

The Loop keyword is missing from a Do statement:

```
Dim i as Integer
Do until i=5
 Beep
 i=i+1
//missing Loop statement
```

A Loop keyword was used without a preceding (matching) Do statement

```
Dim x as Integer
// Do statement missing
 x=x+1
Loop Until x<100
```

The End Select keyword is missing from a Select Case statement:

```
Select case x
case 1
    MsgBox "The party of the first part."
case 2
    MsgBox "The party of the second part."
//no matching End Select statement
```

The Select Case statement is missing:

```
Dim Day as String
case 1 //Select Case statement missing
  Day = "Monday"
case 2
  Day = "Tuesday"
end select
```

The Sub and Function statements cannot appear inside a method.

```
Sub myMethod(x as Integer,y as Integer)
 Sub myMethod (x as Integer,y as Integer)
    .
 end sub
end sub
```

A comma indicates that the second, required parameter is missing:

```
ListBox1.insertRow  2,
```

Using an equals sign when it is not necessary.

```
Return=x*y //equals sign should be a space
```

A keyword was misspelled:

```
If dayNum=1 then
 MsgBox "It's Monday"
end fi  //should be end if
```

System Object

The properties of a **System** object are used to get information about the user's computer.

Super Class

Object

Properties

Name	Type	Description
Keychain	Keychain	Returns the default Keychain. You can assign the default Keychain by setting this to another Keychain. If no Keychain exists, it will try to create one. If you don't want it to create one, you should first call System.KeyChainCount and check against 0.
KeyChainCount	Integer	Returns the number of available Keychains. This number includes all Keychains within the Keychains folder as well as any other Keychains known to the Keychain's manager.
Keyscript	Integer	Allows you to get and set the keyboard script. The values are the values for script systems, e.g., 0 for MacRoman, 1 for MacJapanese, etc. In addition, you can assign negative numbers to get various special effects, e.g., assign -1 to switch to the next layout in the Keyboard menu. For more information, see: http://developer.apple.com/techpubs/mac/Text/Text-386.html
MouseX	Integer	The X coordinate of the mouse (pixels). Measured from the top-left corner of the screen.
MouseY	Integer	The Y coordinate of the mouse (pixels). Measured from the top-left corner of the screen.
MouseDown	Boolean	True if the mouse is down.
PPPStatus	Integer	Current status of the PPP connection. Please refer to the table in the section "PPPStatus values" for a description of the meaning of the values returned by PPPStatus.

Name	Type	Description
QuickTime	QuickTime	Provides access to the QuickTime object which returns certain properties of the QuickTime software installed on the user's computer.
SerialPortCount	Integer	Number of serial ports available.
SerialPort	SerialPort	Read-only array of serial ports on the user's machine.

Methods

Name	Parameters	Description
Gestalt	Code as String ByRef Result as Integer	Returns characteristic of system specified by *Code*. Values of Code are found in *Inside Macintosh*. Returns a Boolean, which is True if successful. Macintosh only.
Pixel	X as Integer, Y as Integer	Returns the color of a screen pixel (as Color), specified in global coordinates. If the specified coordinates are not on any screen, then the color returned is black.
PPPConnect	[UserIntervention as Boolean]	Connects a dialup connection. The optional parameter gives you the option of allowing user intervention during the dial-up process. Some of your users may have more than one ISP they can dial, and the default may not always be who they want to call. If this is the case, set the optional parameter to True to allow user intervention. This feature is not available on Macintosh, as there are no standard dialogs provided for this feature. If the user is running Windows 95/98/ME, or you pass False or omit the parameter, the system will attempt to connect using the first phonebook entry it can find.
PPPDisconnect		Disconnects a PPP connection.

PPPStatus values

The following table summarizes the values returned by the PPPStatus property.

Value	Description
0	There is no connection present. This means that you haven't called the PPPConnect method or the connection process failed. It could also mean that you have called the PPPDisconnect method and the connection has been disconnected.
1	Not used.
3	The connection is being closed. This means that you have called the PPPDisconnect method and the disconnection process has begun. It does not mean that the disconnect is complete.
4	A connection is being attempted. This does not mean that you have a valid internet connection. Calling your TCPSocket or UDPSocket code before retesting for a valid connection may cause an error.
5	You have a valid connection. This means that you can execute your TCPSocket or UDPSocket code because you are fully connected.

Notes

MouseX and MouseY return coordinates with respect to the top-left corner of the user's monitor. The Window and Control classes contain MouseX and MouseY properties that are referenced to the window (in the case of the Control class, the control's parent window).

The SerialPort and SerialPortCount properties work normally in carbonized applications running in Mac OS X. However, carbon applications running under Mac OS "classic" versions cannot access the serial ports. For those machines, build a separate "classic" version of your application.

The Point-to-Point protocol (PPP) is the protocol used for establishing an internet connection via a dial-up modem. Once a connection is established, you can use objects derived from the TCPSocket and/or UDPSocket classes to manage the connection.

In version 5.0, the PPPStatus property and the PPPConnect and PPPDisconnect methods were moved from the Socket class to the **System** object. This is because the 'scope' of PPPStatus, PPPDisconnect, and PPPConnect is system-wide. They apply to all such connections made on the user's computer. For example, calling the PPPDisconnect method will disconnect the connection to the internet, regardless of which application initiated it.

Example

The following displays the number of serial ports on the user's machine and their names. On a 'normal' Macintosh, serialport zero is the Modem port and serialport 1 is the Printer port:

```
EditField1.text=Str(System.serialportcount)
EditField2.text=System.serialport(0).Name
EditField3.text=System.serialport(1).Name
```

The following returns the version of QuickTime installed on the user's computer:

```
Dim s as String
Dim b as Boolean
Dim i,resp as Integer
b=System.gestalt("qtim",resp)
If b then
 s=Hex(resp)
 For i =Len(s) downto 1
  s=Left(s,i)+"."+Mid(s,i+1)
 Next
 MsgBox "QuickTime version "+s
End if
```

The result is:

This example checks whether the application is running on Mac OS 8.5 or higher:

```
// check the system version, since these
//functions are only available
// in MacOS 8.5 and higher
Dim SysVersion as Integer
If System.Gestalt("sysv", sysVersion) and sysVersion >= &h0850 then
// do something cool here
end if
```

To determine whether the application is running under Mac OS X, use the same code but with a constant of &h1000.

See Also

SerialPort, Application classes; App object.

SystemFolder Function

Used to access the System folder. On Windows, it accesses the WINDOWS/SYSTEM directory.

Syntax

***result*=SystemFolder**

Part	Type	Description
result	FolderItem	A FolderItem that represents the System Folder.

Notes

Use the **SystemFolder** function to access the user's System Folder.

The **SystemFolder** function provides a way to access the System folder that will work under different language systems, and will continue to work even if Apple reorganizes the System folder.

If your application will run under Windows, you can first test the TargetWin32 global Boolean constant to determine whether the application is running on a Windows machine.

Examples

See the AppleMenuFolder function for an example.

See Also

AppleMenuFolder, ApplicationSupportFolder, ControlPanelsFolder, DesktopFolder, ExtensionsFolder, FontsFolder, PreferencesFolder, StartupItemsFolder, TemporaryFolder, TrashFolder functions.

TabPanel Control

The standard tab control.

Super Class

PagePanel

Because this is a RectControl, see the RectControl for other properties and events that are common to all RectControl objects.

Properties

Name	Type	Description
Facings	Integer	Facings 0 - North 1 - South 2 - East 3 - West (Mac OS 8.5 and above only)
SmallTabs	Boolean	If True, decreases the vertical dimension of the tab panel (on North/South tabs). See the Notes for an illustration.

Methods

Name	Parameters	Description
Caption	Index as Integer	Allows you to set the caption of the specified tab at runtime. The first tab is numbered zero.

Events

Name	Parameters	Description
Change		The frontmost tab has changed.
GotFocus		(Win32 only) The TabPanel has received the focus. If the user selected the control by tabbing into it, the frontmost tab has a selection marquee.
LostFocus		(Win32 only) The TabPanel has lost the focus.

Name	Parameters	Description
MouseDown	x as Integer y as Integer	The mouse button was pressed inside the TabPanel at the location passed in to x,y. Returns a Boolean. Return True if you are going to handle the MouseDown. The coordinates x, and y are local to the control. Point 0,0 is the top-left corner of the entire control (to the left of the label and above the box that is drawn).
MouseDrag	x as Integer y as Integer	The mouse button was pressed inside the TabPanel and moved (dragged) at the location local to the control passed in to x,y. This event will not occur unless you return True in the MouseDown event.
MouseUp	x as Integer y as Integer	The mouse button was released inside the TabPanel at the location passed in to x,y. This event will not occur unless you return True in the MouseDown event.

Notes

Adding, Labelling, and Renaming Tabs

You modify the tabs in a TabPanel by clicking on the last tab, labelled "…". This displays the Tab Panel Editor, shown below.

From this screen, you can:

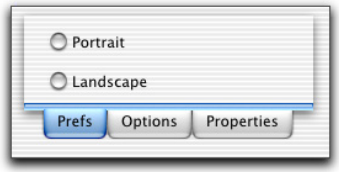
- Rename the first tab label by highlighting it and clicking Edit.
- Add tabs and their labels by clicking the Add button.
- Delete any tab by highlighting it and clicking Delete.
- Rearrange the tabs by highlighting a tab and clicking the Up or Down buttons.

You add controls to a particular panel by clicking its tab and dragging controls to that panel.

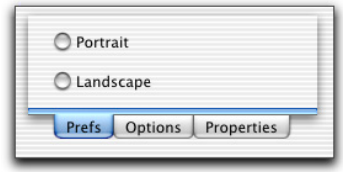
Facings and SmallTabs

The Facings property lets you place the tabs on any side of the TabPanel. The SmallTabs property reduces the vertical (North or South) or horizontal (East or West) size of the tabs.

The following comparison illustrates the effect of the SmallTabs property on a TabPanel that uses the South facing:



SmallTabs Off



SmallTabs On

Placing Controls on Tab Panels

You can place other controls on the surface of any tab panel. Such controls are visible only when the user clicks on that panel's tab to bring it to the front. You can use the TabPanelIndex property of the Control class to read the panel on which the control is located. You cannot move a control to another panel via code. Please note that TabPanelIndex may be superseded by another way of reading a control's 'parent' object and code that uses TabPanelIndex may have to be rewritten.

If the Caption property contains an ampersand character, it will display only if it is preceded by another ampersand character. This is done to make Macintosh and Windows applications behave consistently.

See Also

RectControl class; PagePanel control.

Tan Function

Returns the tangent of the angle specified.

Syntax

***result*=Tan (*value*)**

Part	Type	Description
result	Double	The tangent of *value*.
value	Double	The value (in radians) you want the tangent of.

Notes

The **Tan** function returns the tangent of the angle (in radians) passed to it. If the angle is in degrees, multiply it by PI/180 to convert it to radians.

Examples

This example uses the **Tan** function to return the tangent of a number.

```
Dim d As Double
Const PI=3.14159265358979323846264338327950
d=Tan(45*PI/180) //returns 1.0
```

See Also Atan function.

Target68K Constant

Used to indicate that you are compiling 68K code. Obsolete; REALbasic 4.0 (and above) does not compile 68K code. Retained only for compatibility with previous releases of REALbasic. The final version of REALbasic that compiles 68K code is 3.5.2.

Syntax ***result*=Target68K**

Part	Type	Description
result	Boolean	Returns True if you are compiling 68K code, either as a standalone application or within REALbasic.

Note If the application was compiled using the 68K option only (REALbasic versions 3.5.x and earlier) and run on a PowerPC computer, **Target68K** will return True because 68K code is being compiled.

Target68K, TargetPPC, and TargetCarbon are mutually exclusive subsets of TargetMacOS. TargetPPC is True only if PPC code is running on a PowerPC under a 'classic' Mac OS.

See Also TargetWin32, DebugBuild, TargetMacOS, TargetPPC, TargetCarbon, RBVersion, RBVersionString functions; #If statement.

TargetCarbon Constant

Used to indicate that you are compiling Carbon code.

Syntax ***result*=TargetCarbon**

Part	Type	Description
result	Boolean	Returns True if are compiling Carbon code, either as a standalone application or within REALbasic.

Notes The Carbon version of REALbasic compiles as Carbon while running in the IDE.

Target68K, TargetPPC, and **TargetCarbon** are mutually exclusive subsets of TargetMacOS. TargetPPC is True only if PPC code is running on a PowerPC under a 'classic' Mac OS.

See Also TargetWin32, DebugBuild, TargetMacOS, TargetPPC, Target68K, RBVersion, RBVersionString functions; #If statement.

TargetMacOS Constant

Used to indicate that you are compiling code for any Macintosh OS, 68K or PPC "classic" or Carbon/Mac OS X.

Syntax ***result*=TargetMacOS**

Part	Type	Description
result	Boolean	Returns True if you are compiling code for any Mac OS, either as a standalone application or within REALbasic.

Notes Target68K, TargetPPC, and TargetCarbon are mutually exclusive subsets of **TargetMacOS**. TargetPPC is True only if PPC code is running on a PowerPC under a 'classic' Mac OS.

The final version of REALbasic that compiled 68K code is 3.5.2.

See Also TargetWin32, DebugBuild, Target68K, TargetPPC, TargetCarbon, RBVersion, RBVersionString, functions; #If statement.

TargetPPC Constant

Used to indicate that you are compiling PPC code.

Syntax ***result*=TargetPPC**

Part	Type	Description
result	Boolean	Returns True if you are compiling PPC code, either as a standalone application or within REALbasic.

Notes Target68K, **TargetPPC**, and TargetCarbon are mutually exclusive subsets of TargetMacOS. **TargetPPC** is True only if PPC code is running on a PowerPC under a 'classic' Mac OS.

See Also TargetWin32, DebugBuild, TargetMacOS, Target68K, TargetCarbon, RBVersion, RBVersionString functions; #If statement.

TargetWin32 Constant

Used to indicate that you are compiling code for a Win32 OS.

Syntax

***result*=TargetWin32**

Part	Type	Description
result	Boolean	Returns True if you are compiling Windows code.

See Also

TargetMacOS, DebugBuild, Target68K, TargetPPC, TargetCarbon, RBVersion, RBVersionString functions; #If statement.

TCPIP4DServer Class

Use to open a 4th Dimension/4D Server data source. Please use this class instead of Open4DDatabaseByTCPIP.

Super Class

Database

Properties

Name	Type	Description
Task	String	Description of the connection.

Notes

In order to use this class, you must install in your plugins folder the database plug-in specific to the version of 4D that you want to access. The version number is appended to the classname, e.g., "TCPIP4DServer60".

If you use a 4D data source on Windows, you should copy the 4DOpen.DLL file (provided by 4D) into the same directory as your application. This will ensure that the application can find this library and that the proper version of the library is used.

The set of database plug-ins is included on the REALbasic CD, but you may find more recent versions at the REAL Software web site, www.realsoftware.com.

Example

The following code makes a connection to a 4D Server via TCP/IP.

```
Dim db as TCPIP4DServer67
.
db = New TCPIP4DServer67
db.host = "127.0.0.1"
db.username = "me"
db.password = "test"
.
If db.connect then
//proceed with database operations.
else
//error connecting
end if
```

See Also

ADSP4DServer, Database classes; Open4DDatabaseByADSP, Open4DDatabaseByTCPIP functions.

TCPSocket Control

Used for TCP/IP communication. This class supersedes the Socket class.

Super Class

SocketCore class.

Properties

Name	Type	Description
Address	String	The TCP/IP address to try to connect to.
BytesAvailable	Integer	The number of bytes of data are available in the internal receive buffer.
PPPStatus	Integer	This property is superseded by the PPPStatus property of the System object.
RemoteAddress	String	The remote address of the remote machine you are connected to. Use this instead of the Address property to determine the address of the machine you are actually connected to.

Events

Name	Parameters	Description
Connected		Occurs when a connection has been established.
Error		An error occurred.

Name	Parameters	Description
SendComplete	UserAborted as Boolean	All of the data in the socket buffer has been written (sent). The *UserAborted* parameter enables you to determine whether the user cancelled the transfer by returning True from the SendProgress event. If the user aborted, this value is True, otherwise it is False.
SendProgress	BytesSent as Integer BytesLeft as Integer	Occurs when your network provider queues your data in 'chunks' and is about to send the next chunk. The parameters indicate the amount of progress that has been made during the send. Returns a Boolean. Returning True from this event causes the send to be aborted.

Methods

Name	Parameters	Description
Disconnect		Disconnects the socket, resets it, and issues a Socket 102 error to let you know that the socket has been disconnected.
Listen		Attempts to listen for incoming connections on the currently specified port.
Lookahead	[Encoding as TextEncoding]	Returns a String, containing the data that is available in the internal queue without removing it. The optional *Encoding* parameter enables you to specify the text encoding of the data to be returned.
PPPConnect		Superseded by the PPPConnect method of the System object.
PPPDisconnect		Superseded by the PPPDisconnect method of the System object.
Read	Bytes as Integer, [Encoding as TextEncoding]	Reads the amount of bytes specified from the internal receive buffer. The optional *Encoding* parameter enables you to specify the text encoding of the data to be returned.
ReadAll	[Encoding as TextEncoding]	Reads all the data from the internal buffer. The optional *Encoding* parameter enables you to specify the text encoding of the data to be returned.
Write	Data as String	Writes out *Data* to the remote connection.

Notes

The *Transmission Control Protocol*, or TCP, is the basis for most internet traffic. It is a connection-oriented protocol that provides a reliable way to transfer data across a network.

To establish a connection between two computers using TCPSockets, one computer must be set up to listen on a specific port. The other computer (called the client) then attempts to connect by specifying the network address (or IP address) of the remote machine and the port on which to attempt the connection. In order to send and receive data with a remote machine, both machines must have some indication that this

connection will be established.That happens by either picking a well-defined port for the listener (or server) to listen on, or by some prior arrangement (e.g., you are the author of both the server and the client program).

When a server receives a connection attempt on the port it is listening on, it accepts the incoming connection, and sends an acknowledgement back to the remote machine. Once both machines have reached an agreement (or are "connected"), then you can begin sending and receiving data. When you close your connection with the remote machine, there is a similar handshake process that goes on, so both computers know that the connection is being terminated.

Because of the amount of error checking and handshakes, TCP is an extremely reliable protocol. When you send a packet of information, it is guaranteed to make it to the remote machine unless you have been disconnected, either abortive or orderly. But this feature comes at the cost of high overhead. A typical TCP packet that is sent over the network has around a 40-byte header that goes with it. This header is checked, and changed by all the various machines en route to its destination. This overhead makes TCP a slower protocol to use. The trade-off is speed versus security, in favor of security. If it is speed you are looking for (for example to support a networked game), then you should look into the UDP protocol.

TCPSocket send speeds are faster than the Socket class. Due to the internal loops automatically tightening, and sending the maximum amount of data that the underlying protocols allow per send, you should see an increase in the speed of your calls to the Write method. This new feature is a Macintosh-only feature.

The **TCPSocket** more gracefully handles the disconnection process. It tries to do an orderly disconnect when possible (which allows for all data transfers to finish), and will only fall back on an abortive disconnect when it is not possible to do an orderly one. **TCPSockets** now handle flow control, so sending large amount of data will not drop data during the send or the receive process.

It is possible for packets of data to come in before the socket has finished the connection process. If this happens, the **TCPSocket** is prepared to handle this, so your initial packets will not be lost. This new feature is a Macintosh-only feature; Windows sockets were already prepared to handle these situations.

The SendProgress event allows you to determine send speeds, and tells you how many bytes of data you have sent since your last SendProgress event, as well as how many bytes are left to send. If you return True from the send progress event, you cancel the current transfer. This does not close the socket's connection; it only clears the send buffer. You can use this event to determine if a connection is too slow, and cancel it. Once all of the data has been transferred, you will get a last SendProgress event, followed by a SendComplete event.

The RemoteAddress property for a connecting socket returns the IP address of the remote host connection for Windows sockets. In previous versions of REALbasic, a connecting socket on Windows would return a blank string if you asked it for the RemoteAddress property. One useful thing you can do with the RemoteAddress property is to test to make sure the IP address you wanted to connect to is the same as the IP address you are currently connected to.

In version 5.0 (and above) **TCPSockets** can now be orphaned. An orphaned socket will keep a lock on itself after you call either the Connect or Listen methods. The socket gets unlocked when it has been disconnected. This means that an orphaned socket will continue to work so long as it is connected. This also means that you need to call the Close method before its lock will be released.

On Mac OS X, a user who is not running with root privileges cannot use a port below 1024. You can use a socket to connect to a server on ports less than 1024 but you cannot write a Mac OS X application that will listen on those ports without being root.

See Also

SocketCore, SSLSocket, ServerSocket, UDPSocket classes.

TemporaryFolder Function

Used to access the Temporary folder. On Windows, it accesses the Windows/TEMP directory.

Syntax

***result*=TemporaryFolder**

Part	Type	Description
result	FolderItem	A FolderItem that represents the Temporary folder.

Notes

The Temporary folder is an invisible folder on the user's startup volume. Use it when you need to store a file in a place that the user can't see on the desktop. GetTemporaryFolderItem creates a FolderItem in this folder.

The **TemporaryFolder** function provides a way to access the Temporary folder that will work under different language systems.

If your application will run under Windows, you can first test the global Boolean TargetWin32 to determine whether the application is running in a Windows environment.

Examples

See the AppleMenuFolder function for an example.

See Also

AppleMenuFolder, ApplicationSupportFolder, ControlPanelsFolder, DesktopFolder, ExtensionsFolder, FontsFolder, PreferencesFolder, StartupItemsFolder, SystemFolder, TrashFolder, GetTemporaryFolderItem functions.

TextColor Function

The currently selected Appearance Manager text color.

Syntax

***result*=TextColor**

Part	Type	Description
result	Color	The color used for drawing text.

Notes

This value is useful when you are using Canvas controls to create custom controls that you want to be Appearance Manager savvy. When drawing objects, use this color as the ForeColor value when drawing text.

This value can be changed by the user on the fly using the Kaleidoscope extension so you should access this value during your Paint event handler rather than storing the value.

See Also

DarkBevelColor function, DarkTingeColor function, FillColor function, FrameColor function, HighlightColor function, LightBevelColor function, LightTingeColor function; Color data type.

TextConverter Class

Used to convert text from one encoding system to another.

Methods

Name	Parameters	Description
Clear		Clears the state of the TextConverter object for converting a new string
Convert	InputText as String	Converts Input text

Example

The following example converts the text in an EditField:

```
Dim c as TextConverter
c=GetTextConverter(GetTextEncoding(&h500), GetTextEncoding(0))
EditField2.text=c.convert(EditField1.text)
```

See Also Appendix A and GetTextConverter function.

TextEncoding Class

Used in text conversion.

Properties

Name	Type	Description
Base	Integer	The type of encoding.
Code	Integer	(Mac OS) The Mac OS **TextEncoding** value, useful for declares. You can also use it to compare two encodings: If the Code properties of two TextEncoding objects are equal, then they represent the same text encoding (including base, variation, and format.
Variant	Integer	Specific variation of *Base*.
Format	Integer	Format—only used by Unicode for defining which format of Unicode you wish to use.
InternetName	String	Internet Text Encoding name.

Methods

Name	Parameters	Description
Chr	CodePoint as Integer	Returns a character in the given encoding. The "code point" of the character you want is the same as the value that Asc returns. For example, to get a string that represents the Apple logo in UTF-8, you would use: s = Encodings.UTF8.Chr(&hF8FF)
Equals	otherEncoding as **TextEncoding**	Compares the given encoding to the passed encoding. Returns a Boolean.

Notes

You will find text encoding helpful if you develop:

- Internet applications, such as web browsers or e-mail applications
- Applications that transfer text across different platforms
- Applications based in Unicode

Example

See the example for GetTextConverter.

See Also

Appendix A, GetTextEncoding function, GetInternetTextEncoding function, GetTextConverter function.

TextInputStream Class

In order to read text from a file, you need to create a **TextInputStream** object. **TextInputStreams** have methods that allow to read from a file, check to see if you are at the end of the file, and close the file when you are done reading from it. They are created by calling the OpenAsTextFile method of a FolderItem object.

Super Class

Object

Properties

Name	Type	Description
Encoding	TextEncoding	Specifies the encoding to be defined for a string returned by ReadLIne or ReadAll. It does not actually convert the bytes, but only assigns them an encoding, as if you had called DefineEncoding. It defaults to UTF-8, but you can assign a different encoding to match your file, or even assign Nil if for some reason you want the strings to have an undefined encoding.
EOF	Boolean	The stream has reached the end of the file.

Methods

Name	Parameters	Description
Close		Closes the file.
ReadAll		Returns all of the text (as a string) from the TextInputStream.
ReadLine		Returns the next line of text (as a string) from the TextInputstream.

Examples

This example reads the first line of text from a text file called "MyAppPrefs" in the Preferences folder in the System folder and places the text in an EditField.

```
Dim folder, file As FolderItem
Dim path As String
Dim fileReadFrom As TextInputStream
If TargetMacOS then
 folder=PreferencesFolder
 path=folder.AbsolutePath+"MyAppPrefs"
 file=GetFolderItem(path)
 If file <> Nil then
  fileReadFrom=file.OpenAsTextFile
  EditField1.text=fileReadFrom.ReadLine
  fileReadFrom.Close
 End if
End if
```

This example reads the rows and columns of data from a tab-delimited text file into a ListBox:

```
Dim f As FolderItem
Dim textInput As TextInputStream
Dim rowFromFile,oneCell As String
Dim i As Integer

f=GetOpenfolderItem("text/plain")

If f <> Nil Then
 textInput = f.OpenAsTextFile
 Do
  rowFromFile=textInput.ReadLine
  If ListBox1.ColumnCount < CountFields(rowFromFile,Chr(9)) Then
   ListBox1.ColumnCount=CountFields(rowFromFile,Chr(9))
  End If
 ListBox1.AddRow nthField(rowFromFile,Chr(9),1)
  For i=1 to CountFields(rowFromFile,Chr(9))
   oneCell=NthField(rowFromFile,Chr(9),i)
   ListBox1.Cell(ListBox1.ListCount-1,i-1)=oneCell
  Next
 Loop Until textInput.EOF
 textInput.Close
End If
```

See Also BinaryStream, TextOutputStream classes.

TextOutputStream Class

In order to write text to a file, you need to create a TextOutputStream object. TextOutputStreams have methods that allow to write to a file and close the file when you are done writing to it. They are created by calling the CreateTextFile and AppendToTextFile methods of a FolderItem object.

Super Class Object

Properties

Name	Type	Description
Delimiter	String	The character used to mark the end of a line of text written to the file. The default is a carriage return.

Methods

Name	Parameters	Description
Close		Closes the file.
Write	Text as String	Writes the text passed to the textOutputStream.
WriteLine	Text as String	Writes the text passed to the textOutputStream and appends the Delimiter to the end of the line.

Example

This example displays the Save As dialog box. A text file is then created and the text properties of three EditFields are written to the new file. Finally the file is closed.

```
Dim file As FolderItem
Dim fileStream As TextOutputStream
file=GetSaveFolderItem("application/text","My Info")
fileStream=file.CreateTextFile
fileStream.WriteLine namefield.text
fileStream.WriteLine addressfield.text
fileStream.WriteLine phonefield.text
fileStream.Close
```

See Also

BinaryStream, TextInputStream, FolderItem classes.

The Counter variable and the Variable Following Next Must be the Same Error

In a For loop, if you use both a counter variable and a variable in the Next statement, they must match. The variable on the Next statement can be omitted.

Example

In the following For loop, the counter variable is i, so the variable on the Next statement must also be i.

```
Dim i,j as Integer
For i=1 to 10
 j=j+1
Next j
```

See Also

For statement.

The Size of an Array Must Be a Constant or a Number Error

When you declare and size an array using a Dim statement, the size of the array must be indicated by either a constant or a number.

Example Trying to size an array using a variable:

```
Dim j as Integer
j=5
Dim a(j) as Integer
```

See Also Dim statement.

There is no Class with this Name Error

You used a nonexistent class name.

Examples Using nonexistent class names on a Dim statement.

```
Dim Fred as Husband //'Husband' not a data type
Dim f as Folder  //should be FolderItem
Dim d as longdouble //no such thing; programmer is lost
```

Using a nonexistent class with a New operator.

```
Dim f as FolderItem
f=New Folder
```

See Also Dim statement, New operator.

This Array Has Fewer Dimensions than You Have Provided Error

You tried to execute a built-in array method that requires a one-dimensional array on a multi-dimensional array.

Example

You can't sort a two-dimensional array.

```
Dim aInts(3,3) as Integer
Dim i,j as Integer
For i=0 to 2
 For j=0 to 2
  aInts(i,j)=i*j
 Next
Next
aInts.Sort //doesn't work on 2D arrays
```

Using syntax that indicates that you are referencing an element in a two-dimensional array when it has been declared as a one-dimensional array:

```
Dim aInts(3) as Integer
aInts(0,0)=3 //incorrect
```

Trying to change a one-dimensional array to a two-dimensional array with a Redim statement:

```
Dim aInts(3) as Integer

Redim aInts(5,5)
```

See Also

Append, Insert, Remove, Sort, Dim, Redim methods.

This Array Has More Dimensions than You Have Provided Error

In referring to an element of a multidimensional array, you didn't provide subscripts for all the dimensions.

Example

```
Dim a(5,4) as String
a(1)="Bob"
```

See Also

Dim statement.

This #else Does Not Have a Matching #If Preceding It Error

An extra #else keyword was used in an #If statement (conditional compilation) or an #else statement is used where no #if preceded it.

Example An #else keyword was used instead of an Else keyword in an If statement.

```
Dim theNumber As Integer
Dim digits As Integer
theNumber=33
If theNumber<10 Then
  digits=1
#else //should be else
  digits=3
End If
```

See Also #If, If statements.

This #endif Does Not Have a Matching #If Preceding It Error

A #If statement was omitted from an #if…#endif structure.

Example The If statement in this example should be #If.

```
If TargetMacOS then
 Beep
#endif
```

See Also #If statement.

This Class is Missing One or More Methods of an Interface it Inherits Error

One or more of the methods of this interface could not be found in this class, even though it claimed to implement the interface.

This Global Variable has the Same Name as a Class Error

You created a global variable in a module that uses the same name as one of the classes in your project.

This Global Variable has the Same Name as a Global Function Error

You reused the name of a method or function in a module as the name of a property in a module.

Example Defining 'myAlert' as a method in a module and also defining 'myAlert' as a property in the module.

This is not An Array but you are using it as One Error

You passed an object to a method or function that expects an array but did not get one. It also occurs if you are trying to assign a value to an element of an alleged array but it is not an array.

Examples

```
Dim aInts(4) as Integer
Dim f as FolderItem
Redim f(10) //not an array


Dim a as String
a(1)="Ted" //not an array
```

See Also Dim statement, Redim operator.

This Item Conflicts with Another Item of the Same Name Error

Occurs when you are calling an overloaded method and REALbasic cannot determine which version you meant to call. This happens when two methods have the same name, number of parameters, and the data types of the parameters match.

This error will also occur if a method name conflicts with another type of project item such as a window, menuitem, control, and so forth. For example, if you have a global function named Foo and a window called Foo.

Example

The method myOverLoadedMethod takes three integer parameters, but it is defined twice (performing different functions) in a window's Code Editor. A call to myOverLoadedMethod produces the error. The solution is to rename or eliminate the second instance of myOverLoadedMethod.

Instance 1

```
Sub myOverLoadedMethod(a as Integer, b as Integer, c as Integer)
 Dim d as Integer
 d=a*b/c
 MsgBox str(d)
End Sub
```

Instance2

```
Sub myOverLoadedMethod(x as Integer, y as Integer, zas Integer)
 Dim d as Integer
 d=x/z*y
 MsgBox str(d)
End Sub
```

Calling method

```
Dim a,b,c as Integer
a=5
b=10
c=15
myOverloadedMethod(a,b,c)
```

This Kind of Array Cannot be Sorted Error

You tried to sort a Boolean array.

Example

You tried to sort an array of objects instead of an array of alphas or numbers. For example, an array of Date objects cannot be sorted, but an array of TotalSeconds properties of Date objects can be sorted, since the TotalSeconds property is a Double.

This example array cannot be sorted.

```
Dim d(2) as Date
d(0)=New Date
d(1)=New Date
d(2)=New Date
d(0).year=1965
d(1).year=1970
d(2).year=1950
d.Sort
```

The last line of this example produces the error message.

```
Dim theTruth(3) as Boolean
theTruth(0)=True
theTruth(1)=True
theTruth(2)=False
theTruth(3)=False
theTruth.Sort
```

See Also

Sort method.

This Line is too Complex Error

You wrote a line of code that REALbasic couldn't understand. You need to simplify the line.

This Local Variable or Constant has the Same Name as a Declare in this Method Error

You used the name of a local variable or local constant in a Declare statement in a method.

See Also Declare statement.

This Local Variable or Parameter has the Same Name as a Constant Error

You declared a local variable in a method or a parameter in a Sub or Function that has the same name as a constant.

Example You cannot use the same name for a constant and a local variable:

```
Const Version="10B"
Dim Version as String
```

See Also Const, Dim statements.

This Method Doesn't Return a Value Error

You used a method in an expression as though it returned a value, but it doesn't.

Example The user-written method myColor is not a function, so this syntax in a calling method is not correct:

```
Dim c as Color
c=myColor //does not return a color
```

This Method is Protected. It can only be Called From Within its Class Error

You tried to access a protected method from outside the class that owns it. A protected method is accessible only from within its parent object—its class, window, or module.

You can unprotect a method by unchecking the Protected checkbox in the Method Declaration dialog box.

This Method, Property or Variable Does Not Exist Error

You used a syntax that implies that a term is a method, property, or variable, but it does not exist. It also occurs when you use a variable in an assignment statement but omit the equals sign, making it appear that it is a method or property.

Examples

Omitting an equals sign:

```
Dim myString as String
myString "Columbus"
```

Calling a method that has not been defined:

```
myCustomMethod(7,5,3)
Dim aFiles(3) as FolderItem
aFiles.List //no such thing
```

Assigning a value to a variable that has not been declared using a Dim statement:

```
i=10
```

This is the Action event handler of a PushButton; the user wants to display the caption property of the PushButton.

```
MsgBox caption //should be me.caption*
```

Trying to set in code a property that can only be set in the Properties window. The following line of code tries to set the value of the Bold property of a MenuItem, but REALbasic does not recognize the Bold property in code.

```
SearchFind.Bold=True
```

Using Me in a method or function that is not attached to an object. The following method tries to return the caption property of the calling PushButton, but this is not permitted.

```
Function myCaption() as String
  Return Me.Caption
End Function
```

You can only access the calling object's properties using Me within an event handler belonging to that object.

Assigning a value to a constant:

```
Const SerialPort=1080
SerialPort=1040
```

The programmer intended to declare aNames as an array in the Dim statement but forgot to do so.

```
Dim aNames as String
aNames.Append "Walter Kerr"
```

This Method Requires Fewer Parameters than were Passed Error

You passed too many parameters to a method. Another possibility is that you passed the required number of parameters but did not separate them with commas.

Example

Passing the required number of parameters but omitting a comma:

```
ListBox1.AddRow  2 "Chicago"
```

Passing too many parameters:

```
listBox1.insertRow 2,"Charlie","Dallas"
```

The user-written function, myFunction, takes two parameters:

```
Dim d as Double
d=myFunction(5,6,8) //correct
```

This Method Requires More Parameters than were Passed Error

You didn't pass all the required parameters to a method. Another possibility is that you passed the required number of parameters but did not separate them with commas.

Example

Passing the required number of parameters but omitting a comma:

```
ListBox1.insertRow  2 "Chicago"
```

Passing too few parameters:

```
ListBox1.insertRow "Chicago"
```

```
Dim c as Color
c=RGB(100,100) //returns "RGB takes 3 parameters"t
```

In the following examples, myFunction takes two parameters and returns a Double:

```
d=myFunction(5,6) //correct
d=myFunction (5,6) //also correct
d=myFunction 5,6 //error
d=myFunction //also an error
```

This Name is Already in Use Error

You used a variable name in a Dim statement that has already been declared in another Dim statement or as a property. Or, you tried to declare a property with the same name as another property.

Example

```
Dim a,b as Double
Dim b as Double //already in use
```

See Also

Dim statement.

This Property is Protected. It can only be Used From Within its Class Error

You tried to access a property from outside the class that owns it. A Protected property can be used only within its parent class, window, or module.

You can unprotect a property by unchecking the Protected checkbox in the Property Declaration dialog box.

This Property Has the Same Name as a Class Method Error

You reused a name of a method in a class as the name of a property in the class.

This Property Has the Same Name as an Event Error

You used the name of an event as the name of a property. You need to rename the property.

Example Using 'Open' as both the name of an event and a property in a class.

This Property Has the Same Name as a Method Error

You reused the name of a method as the name of a property.

Thread Class

Threads are used to execute code in the background, independent of the user interface.

Super Class

Object

Events

Name	Parameters	Description
Run		The code that executes when the Run method is called.

Properties

Name	Type	Description
StackSize	Integer	Size of the stack, in bytes. Can be set when the thread is not running. The default stack size is 64K. It was previously 256K. A stack size of zero indicates the default.
ThreadID	Integer	ID of thread, assigned at runtime.

Methods

Name	Parameters	Description
Run		Executes the thread's Run event handler.

Notes

Thread class objects are used to execute code that needs to run independently of the user interface. For example, if you were drawing a complex image that takes several minutes to render, this code could be executed in a thread so that the user could continue using the user interface without having to wait for the rendering to finish.

A thread can also be used to allow a very time-consuming loop to run in the background. Since unthreaded loops take over the interface, preventing the user from making menu selections or interacting with a window's controls, this is a good way to allow the user to continue working with the application while a lengthy process is taking place. See the For, While, and Do loops.

Threads yield time to other threads and other applications each time they execute a looping construct such as For, While, and Do.

To create a thread, create a new class and make **Thread** its super class. Put the code you wish to execute in the thread's Run event handler. To execute the thread, simply create a new thread object and call its Run method. You can do this by adding the instance of the Thread class to a window, as shown in the example, or instantiate it in the usual way.

Example

The following thread, Calculate, calls a function in a module that does a time-consuming calculation.

```
Dim r as Double
r=Interpolate(1, 100,000,000)
```

To do the calculation, drag the Thread class, Calculate, from the Project Window to a window. This creates an instance of the thread, Calculate1. Add a control to the window that has the following code:

```
Calculate1.run
```

You can obtain the ID of this instance of the thread by reading the value of

```
Calculate1.ThreadID
```

Instead of dragging the Calculate class to a window, you can create the instance via code and call its run method:

```
Dim t as Calculate
t=New Calculate
t.run
```

See Also

Semaphore, CriticalSection classes.

Ticks Function

Returns the number of ticks (60th of a second) that have passed since the user's computer was started.

Syntax

***result*=Ticks**

Part	Type	Description
result	Integer	The number of ticks that have passed since the computer was started.

Examples

This example displays in message box the number of minutes the computer has been on.

```
Dim minutes As Integer
minutes=Ticks/60/60
MsgBox "Your computer has been on for"+Str(minutes)+" minutes."
```

See Also

Microseconds function

Timer Control 

A **Timer** is a non-visible control that can execute code after a specified period of time or at a repeated interval.

Super Class

Object

Properties

Name	Type	Description
Enabled	Boolean	Enables you to turn the Timer on or off. When Enabled is True, the Timer is on.
Index	Integer	The number of the control when it is part of a control array.
Left	Integer	The left side of the control in local coordinates (relative to the window).
Mode	Integer	The interval at which the Action event will be executed. 0=off, 1=single, 2=multiple Default is 2.
Name	String	The name of the object.
Period	Integer	The time (in milliseconds) between executions. Periods of less than or equal to zero default to a value of 1 millisecond. The default value is 1000. The rate that a Timer control can actually fire depends on the speed of the host computer, the operating system, and other tasks the computer is doing. It is possible to set Period to a value that cannot be achieved by the computer that is running the application.
Top	Integer	The top of the control in local coordinates (relative to the window).

Methods

Name	Parameters	Description
Reset		Resets the Timer and restarts it.

Events

Name	Parameters	Description
Action		The code that will execute after the Period specified and at the Mode specified.

Notes

The Mode property controls the interval used to execute the **Timer** control's Action event handler. If the Mode is not 0 (zero), the Timer control waits for the Period to pass

before executing the Action event handler. If the Mode is set to 0 (zero) at Runtime, the **Timer** control will immediately cease waiting and the Action event handler will no longer be executed.

The **Timer** control will continue to execute its Action event handler (assuming the Mode property is not 0) regardless of whether the window is frontmost or not. The visibility of the window also has no impact on the execution of a **Timer** control's Action event handler. The **Timer** has been designed so that it yields time to other applications running on the computer so as not to bog down the machine.

See Also

Object class.

Titlecase Function

Returns the string passed to it with all alphabetic characters in Titlecase.

Syntax

***result*=Titlecase(*value*)**

OR

***result=stringVariable*.Titlecase**

Part	Type	Description
result	String	A copy of the original string with all characters converted to their titlecase equivalent.
value	String	The original string.
stringVariable	String	Any variable of type String.

Notes

Converts all characters in a string to lowercase characters and then converts the first character of each word to uppercase. Numbers are not affected.

Examples

The example below converts the values passed to **Titlecase**.

```
Dim s As String
s=Titlecase("tHe Quick fOX") //returns "The Quick Fox"
s=Titlecase("THE LAZY DOG") //returns "The Lazy Dog"
```

See Also

Lowercase, Uppercase functions.

ToolbarItem Control

Used to create a toolbar in a window. Supported on Mac OS X 10.2 and above only. ToolbarItems can be placed in windows for other platforms for layout purposes only.

Super Class Control

Properties

Name	Type	Description
Caption	String	The text label below the ToolbarItem. See Notes.
ControlOrder	Integer	Determines the left to right placement of the icons on the Toolbar.
Enabled	Boolean	True if the ToolbarItem is enabled.
Handle	Integer	A toolbarItemRef, used in Declare statements, of use only to toolbox programmers.
HelpTag	String	Standard help tag for the item.
HelpTagLong	String	Help tag that appears when the user holds down the Command key.
Image	Picture	The picture to be displayed by the ToolbarItem. If no picture is associated with the ToolbarItem, a "?" icon is displayed.
Left	Integer	The distance (pixels) from the left of the window in the Window Editor. In the runtime application, the ordering of toolbar items is strictly based left to right on control order.
Top	Integer	The distance (pixels) from the top of the window in the Window Editor. In the runtime application, the ordering of toolbar items is strictly based left to right on control order.

Events

Name	Parameters	Description
Action		Occurs when the user clicks on the ToolbarItem.
Close		Occurs when the ToolbarItem is destroyed.
Open		Occurs when the ToolbarItem is first created.

Methods

Name	Parameters	Description
SetImageFromIcon	Creator as String Type as String	Assigns a picture to the Image property using an icon registered with the system using the Type and Creator passed.

The following are popular icons for use with SetImageFromIcon:

Creator	Type	Description
macs	tdel	Delete

Creator	Type	Description
macs	tfav	Favorites
macs	thom	Home
macs	burn	Burn
macs	ejec	Eject
macs	CLIP	Clipboard
macs	desk	Desktop
macs	FNDR	Finder
macs	trsh	Trash
macs	ftrh	Full Trash
macs	docu	Document
macs	gurl	URL
macs	udsk	iDisk
macs	ilht	HTTP
macs	ilft	FTP
macs	gnet	Network

For a complete list of Apple-supplied icons, see Icons.h in the Universal Headers.

Note If the Caption property contains an ampersand character, it will display only if it is preceded by another ampersand character. For example, if you set the Caption property to "Bob & Ray", only "Bob Ray" will display. You need to set it to "Bob && Ray". The ampersand character is used on Windows for keyboard accelerators.

See Also StandardToolbarItem control.

TrashFolder Function

Used to access the items in the Trash Can (aka, "folder"). On Windows, a call to TrashFolder returns a FolderItem that references the Recycle Bin and you can list the items in it.

Syntax ***result*=TrashFolder**

Part	Type	Description
result	FolderItem	A FolderItem that represents the Trash folder or Recycle Bin.

Notes

The Trash folder is represented on the Macintosh desktop by the Trash Can icon. When a user drags items in or out of the Trash Can, they are in fact, dragging those items in or out of the Trash folder.

The **TrashFolder** function provides a way to access the Trash folder that will work under different language systems.

Examples

See the AppleMenuFolder function for an example.

See Also

AppleMenuFolder, ApplicationSupportFolder, ControlPanelsFolder, DesktopFolder, ExtensionsFolder, FontsFolder, PreferencesFolder, StartupItemsFolder, SystemFolder, TemporaryFolder functions.

Trim Function

Returns the string passed with leading and trailing spaces removed.

Syntax

result*=Trim*(SourceString)

OR

***result=stringVariable*.Trim**

Part	Type	Description
result	String	*SourceString* with leading and trailing spaces removed.
SourceString	String	The source, a copy of which, to be returned with leading and trailing spaces removed.
stringVariable	String	Any variable of type String.

Examples

This example removes the spaces from either side of the string passed:

```
Dim s as String
s=Trim("  Hello World  ") //Returns "Hello World"
```

See Also

LTrim, RTrim, Asc, Chr, InStr, Len, Left, Mid, Right functions.

True Function

Used to set a Boolean object to the value of 'True' or test whether an existing Boolean expression is equal to 'True.'

Syntax

***expression*=True**

Expression is any valid Boolean expression.

See Also

False, And, Or, Not functions.

Type Mismatch Error

Occurs when you try to pass a parameter of an incorrect data type or use an incorrect data type in an assignment statement.

Examples

Trying to assign a string to an Integer variable:

```
Dim n as Integer
n="Anthony"
```

Trying to assign a value to an array instead of an element of the array.

```
Dim truth(3) as Boolean
truth=True //must have a subscript
```

TypeMismatchException Error

Occurs when you try to assign a value to an object that is an incorrect data type.

Super Class

RuntimeException

Notes

A **TypeMismatchException** error occurs when you try to assign a value of an incorrect data type to an object. This error can occur only if the compiler cannot determine the type of the value at compile time — for example, when using variants. Ordinarily, the compiler catches incorrect typing when you try to compile the application.

Example

The following code assigns a picture that has been dragged into the Project window to the variant, v, and then tries to assign v to an Integer. The exception handler displays a

message box, allowing the developer to track down the problem. Exception handling prevents the built application from quitting when the error occurs.

```
Dim v as Variant
Dim i as Integer
v=ProductLogo //a picture
i=v
Exception err
If err IsA TypeMismatchException then
 MsgBox "The variable i must be assigned a number!"
end if
```

See Also

RuntimeException class, OutofBoundsException error, IllegalCastException error, StackOverFlowException error, NilObjectException error, Function statement, Raise statement, Nil function, Exception block.

Ubound Function

Returns the index of the last element in an array.

Syntax

***result*=Ubound(*array,[dimension]*)**

Part	Type	Description
result	Integer	The index of the last element in the array specified.
array	Array object	The array whose last element number you want.
dimension	Integer	Optional: Relevant only for multi-dimensional arrays. Used to specify the dimension for which you want the last element. The first dimension is numbered 1. If passed -1, it will return the number of dimensions in the array. If passed a non-existent dimension, it will cause an OutOfBoundsException error.

Notes

The **Ubound** function can be used to determine the last element of an array, but it can also be used to determine the size of an array. It may appear at first that the last element number and the size of the array are one in the same but in fact they are not. All arrays have a zero element. In some cases element zero is used and in other cases it is not. You will need to keep this in mind when using the **Ubound** function to determine the number of values you have in the array. If the array is zero-based, then element zero is used to store a value and you will have to add one to the value returned by the **Ubound** function to make up for it.

Examples

This example replaces each occurrence of X in an array with Y.

```
Dim i As Integer
For i=0 to Ubound(Names)
  If Names(i)="X" Then
    Names(i)="Y"
  End If
Next
```

See Also

Append method, Dim statement, Insert method, Redim method, Remove method, Sort method.

UDPSocket Class

Used for UDP (User Datagram Protocol) communications.

Super Class

SocketCore class.

Properties

Name	Type	Description
BroadcastAddress	String	The machine's broadcast address. You can specify this address in a Write, and the data will be broadcast across the network (but you will not receive the data you sent back).
PacketsAvailable	Integer	The number of packets available in the internal receive buffer.
RouterHops	Integer	Specifies how many router hops the data sent out can make. This is also known as the Time To Live (or TTL). This property only applies to multicast sends.
SendToSelf	Boolean	Specifies whether the data you send out will be sent to yourself as well. This is also known as loopback. This property only applies to multicast sends. Setting the SendToSelf property to True may not work on all versions of Windows. MSDN states that the SendToSelf property on Windows 95 and NT 4 can not be turned off. A multicasting socket will always receive the data it sends out.

Methods

Name	Parameters	Description
JoinMulticast Group	Group as String	Attempts to join the specified multicast group. Returns a Boolean. If the socket successfully joined, it returns True. You can join as many multicast groups as you'd like. The *Group* parameter specifies a Class D IP address in the range: 224.0.0.0 to 239.255.255.255.
LeaveMulticastGroup	Group as String	Leaves the specified multicast group.
Read	Data as DataGram, [Encoding as TextEncoding]	Retrieves a datagram from the internal receive buffer. The address property of the Datagram is the remote address from which the data was sent. The optional *Encoding* parameter enables you to specify the text encoding of the data to be returned.
Write	Data as DataGram	Writes the data to the specified address.
Write	Address as String Data as String	Constructs a Datagram and sends the data to the specified address. If the address is part of a multicast group, all members of the group will get the data. If the address is the broadcast address, everyone on the network will get the data. If the address is an IP address, the data is unicast to that address only.

Notes

The **UDPSocket** does not have an icon of its own in the Controls palette. The easiest way to add it to a window is to display the window's contextual menu by Control-clicking (Macintosh) or right-clicking (Windows) and choosing Add ▶ SocketCore ▶ UDPSocket. You can also drag a TCPSocket control to a window and then use the Properties window to change its SuperClass to UDPSocket. However, you do not need to add a UDPSocket control to a window in order to use it. Since it is not a subclass of Control, you can instantiate it via code.

The User Datagram Protocol, or UDP, is the basis for most high speed, highly distributed network traffic. It is a connectionless protocol that has very low overhead, but is not as secure as TCP. Since there is no connection, you do not need to take nearly as many steps to prepare when you wish to use a UDP socket.

In order to use a **UDPSocket**, it must be bound to a specific port on your machine. This is done using the Connect method of the SocketCore class. Once the bind has occurred, the **UDPSocket** is ready for use. It will immediately begin accepting any data that it sees on the port it has bound to. It also allows you to send data out, as well as set UDP socket options (which will be described later).

In order to differentiate between which machine is sending you what data, a **UDPSocket** uses a data structure known as a Datagram. A Datagram consists of two parts, the IP address of the remote machine that sent you the data, and the payload —

or the actual data itself. When you attempt to send data out, you must specify information in the form of a Datagram. This information is usually the remote address of the machine you want to receive your packet, the port it should be sent to, and the data you wish to send the remote machine.

UDP sockets can operate in various modes, which are all very similar, but have vastly different uses. The mode that most resembles a TCP communication is called "unicasting." This occurs when the IP address you specify when you write data out is that of a single machine. An example would be sending data to www.realsoftware.com, or some other network address. It is a Datagram that has one intended receiver.

The second mode of operation is called "broadcasting." As the name implies, this is a broadcasted write. It is akin to yelling into a megaphone. Everyone gets the message, whether they want to or not. If the machine happens to be listening on the specific port you specified, then it will receive the data on that port. This type of send is very network intensive. As you can imagine, broadcasting can amount to huge amount of network traffic. The good news is, when you broadcast data out, it does not leave your subnet. Basically, a broadcast send will not leave your network to travel out into the world. When you want to broadcast data, instead of sending the data to an IP address of a remote machine, you specify the broadcast address for your machine. This address changes from machine to machine, so REALbasic provides a property of the **UDPSocket** class which tells you the correct broadcast address.

The third mode of operation for UDP sockets is "multicasting." It is a combination of unicasting and broadcasting that is very powerful and practical. Multicasting is a lot like a chat room: you enter the chatroom, and are able to hold conversations with everyone else in the chatroom. When you want to enter the chatroom, you call JoinMulticastGroup, and you only need to specify the group you wish to join. The group parameter is a special kind of IP address, called a *Class D IP*. They range from 224.0.0.0 to 239.255.255.255. Think of the IP address as the name of the chatroom. If you want to start chatting with two other people, all three of you need to call JoinMulticastGroup with the same Class D IP address specified as the group. When you wish to leave the chatroom, you only need to call LeaveMulticastGroup, and again, specify the group you wish to leave. You can join as many multicast groups as you like, you are not limited to just one at a time. When you wish to send data to the multicast group, you only need to specify the multicast group's IP address. All people who have joined the same group as you will receive the message.

Multicasting has some extra features that make it an even more powerful utility for network applications. If you wish to receive the data you sent out with a multicast send, you can set the SendToSelf property (also known as loopback) of the **UDPSocket**. If it is set to True, then when you do a send (to a multicast group) you will get that data back. You can also set the number of router hops a multicast datagram will take (also known as the Time to Live). When your Datagram gets sent out, it runs through a

series of routers on the way to its destinations. Every time the Datagram hits a router, its RouterHops property gets decremented. When that number reaches zero, the Datagram is destroyed. This means you can control who gets your datagrams with a lot more precision. There are some "best guesses" as to what the value of RouterHops should be.

Value	Description
0	Same host
1	Same subnet
32	Same site
64	Same region
128	Same continent
255	Unrestricted

Note that if your Datagram runs through a router that does not support multicasting, it is killed immediately.

Due to the connectionless functionality of UDP, it does not make any guarantees that your data will reach its destination. You can work around this by creating your own protocol on top of the UDP protocol which acknowledges receives.

The UDPsocket operates in asynchronous mode, but the Connect method works synchronously. If the connect fails, you will get an error event immediately, and the IsConnected property will be set immediately.

See Also

SocketCore, Datagram classes.

UnsupportedFormatException Error

Occurs when you use a String expression that cannot be evaluated.

Super Class

RuntimeException

Notes

REALbasic allows you to use a string expression to set the column widths via the Column class or the ColumnWidths property of the ListBox. You can use the percent sign or the "*" symbol, as described in those sections. If you use a character that is not permitted, an **UnsupportedFormatException** will occur.

Example

The following specification causes an **UnsupportedFormatException** runtime error:

```
ListBox1.ColumnWidths="50,50i"
```

See Also RuntimeException class; Exception Block

Uppercase Function

Converts all characters in a string to uppercase characters.

Syntax ***result*=Uppercase(*value*)**

OR

***result=stringVariable*.Uppercase**

Part	Type	Description
result	String	A copy of the original string with all characters converted to their uppercase equivalent.
value	String	The original string.
stringVariable	String	Any variable of type String.

Notes Returns the value with all alphabetic characters in uppercase.

Examples The example below converts the value passed to uppercase.

```
Dim s As String
s=Uppercase("tHe Quick fOX") //returns "THE QUICK FOX"
s=Uppercase("the 5 lazy dogs") //returns "THE 5 LAZY DOGS"
```

See Also Titlecase, Lowercase functions.

UserCancelled Function

Used to determine if the user has pressed Command-period (Macintosh) or Escape (Windows) to cancel the execution of code.

Syntax ***result*=UserCancelled**

Part	Type	Description
result	Boolean	*Result* is True if the user has pressed Command-period (Macintosh) or Escape (Windows) and False if not.

Notes The **UserCancelled** function will continue to return True until the event handler that was executing when the user pressed Command-period/Escape is finished.

Examples

This example uses the **UserCancelled** function to exit a For loop if the user presses Command-period/Escape.

```
Dim i As Integer
For i = 0 to 10000
  ProgressBar1.Value = i
  If UserCancelled Then
    Exit
  End If
Next
```

Val Function

Returns the numeric form of a string.

Syntax

***result*=Val(*string*)**

OR

***result=stringVariable*.Val**

Part	Type	Description
result	Double	The numeric equivalent of the string passed.
string	String	Any valid string expression.
stringVariable	String	Any variable of type String.

Notes

The **Val** function stops reading the string at the first character it doesn't recognize as part of a number. All other characters are automatically stripped.

It does recognize prefixes &0 (octal), &b (binary), and &H (hexadecimal). However, spaces are not allowed in front of the ampersand. That is, " &HFF" does not return a value, but "&H FF" returns 255.

The CDbl function is the same as the **Val** function but is used when you need to pass a string that uses a character other than the period (.) as the decimal separator. It uses the character specified by the Numbers Control panel. **Val** should generally be used to convert internal data, but not data entered by the user. Val is not international-savvy, but CDbl is.

The CStr function is the same as the Str function but is used when you need to pass a number that uses a character other than the period (.) as the decimal separator. It uses the character specified in the Numbers Control panel.

Val returns zero if *string* contains no numbers.

Examples

These examples use the **Val** function to return the numbers contained in a string.

```
Dim n As Integer
n = Val("12345")  //returns 12345
n = Val(" 12345") //returns 12345
n = Val("123 45") //returns 123
n = Val("&HFFF")  //returns 4095
n = Val("&b1111")  //returns 15
```

See Also

Str, CDbl, and CStr functions; & operator.

Variant Data Type

A **Variant** is a special data type that can contain any type of data. Use the VarType function or the Variant class's Type method to determine the data type of a variant.

Super Class

Object

Properties

Name	Type	Description
BooleanValue	Boolean	Returns the value of the variant as a Boolean.
ColorValue	Color	Returns the value of the variant as a Color.
DateValue	Date	Returns the value of the variant as a Date. When retrieving the string value of a date, the string is returned in SQL date format *YYYY-MM-DD HH:MM*.
DoubleValue	Double	Returns the value of the variant as a Double.
IntegerValue	Integer	Returns the value of the variant as an Integer.
ObjectValue	Object	Returns the value of the variant as an Object.
StringValue	String	Returns the value of the variant as a String.

Methods

Name	Parameters	Description
Equals	v as **Variant**	Returns a Boolean. Returns True if the current variant is of the same data type and value as the variant passed as a parameter.
Hash	Integer	Returns a hash value for the variant, i.e., one that will always be the same for any variant with the same value. The Hash value is guaranteed to be unique for integers and colors. For objects, it's guaranteed to be unique among all objects currently in existence (though once an object dies, a future object may use the same hash value).
IsNull		Returns a Boolean. True if the variant is Null.

Name	Parameters	Description
IsNumeric		Returns a Boolean. True if the variant is a numeric data type.
Type		Returns an Integer, indicating the data type of *v*. If the variant is not Nil, it provides the same functionality as the VarType global function. See the table of data types for the VarType function.

Notes

When you assign a variant to a string, numeric, or date variable, REALbasic converts the value of the variant to the variable's data type. For example,

```
Dim v as Variant
Dim n as Integer
Dim s as String
v=25
s=v   // s is "25"
v="25"
n=v // n is 25
```

The Type method returns the type of the variant, provided it is not Nil. For example, the following code generates an NilObjectException error.

```
Dim v as Variant
Dim i as Integer
i=v.type
```

On the other hand, the following example returns 0.

```
Dim v as Variant
Dim i as Integer
i=VarType(v)
```

See Also

VarType function; Dictionary, DatabaseField classes.

VarType Function

Used to determine the data type of a variable.

Syntax

***result*=VarType(*value*)**

Part	Type	Description
result	Integer	Indicates the data type of *value*.
value	Variant	Value to be typed.

Notes

Result takes on the following values:

Result	Description
0	Nil
2	Integer
5	Double or Single
7	Date
8	String
9	Object
11	Boolean
16	Color

Please note that some earlier versions of REALbasic returned 13 for Object.

Example

The following line of code returns 8 (data type of String):

```
EditField1.text=Str(VarType("Herman"))
```

See Also

Variant class, Integer, Single, Double, String, Boolean data types.

Vector3D Class

Used to represent absolute or relative position of a vector an RB3DSpace three-dimensional space.

Properties

Name	Type	Description
X	Double	x coordinate of a point in 3D space.
Y	Double	y-coordinate of a point in 3D space.
Y	Double	z-coordinate of a point in 3D space.

Methods

Name	Parameters	Description
Add	vec as **Vector3D**	Adds the vector *vec* to the current vector, storing the result in this object.
Copy	vec as **Vector3D**	Makes this **Vector3D** match the passed vector.
Cross	vec as **Vector3D**	Returns the cross-product of two vectors. v1.Cross(v2) gives you a new vector which is perpendicular to the plane defined by v1 and v2. Returns a **Vector3D**.

Name	Parameters	Description
Dot	vec as **Vector3D**	Returns the dot-product of two vectors. v1.Dot(v2) is proportional to the cosine of the angle between v1 and v2. Returns a Double.
Length		Returns the length of the vector in 3D space. Returns a Double.
LenSquared		Returns the squared length of the vector in 3D space. Returns a Double.
Minus	vec as **Vector3D**	Returns the difference of this vector and the vector *vec*.
MultiplyBy	d as Double	Multiplies this vector by the scalar *d*, storing the result in this object. This scales the vector; another way to scale a vector is to assign to its Length or LenSquared property.
Normalize		Scales the vector so that its square length (and length) is 1.
Plus	vec as **Vector3D**	Returns a **Vector3D**. Returns the sum of this vector and the passed vector.
Subtract	vec as **Vector3D**	Subtracts the passed vector from this one, storing the result in this object.
Times	d as Double	Returns a **Vector3D**. Returns the product of this vector and the scalar, *d*.

Constructor Syntaxes

Syntax	Parameters	Description
New **Vector3D**(x, y, z)	x as Double y as Double z as Double	Creates a **Vector3D** and initializes x, y, and z to the values passed.
New **Vector3D**(vec)	vec as **Vector3D**	Creates a **Vector3D** and initializes the new vector to the same value as the one given.

Notes

The **Vector3D** class is used to represent absolute or relative position. It's simply the X, Y, and Z coordinates of a point in 3D space. Depending on how you intend to use it, they can either mean the absolute position of something in 3D space, or a change in position. You can get its length (i.e., its distance from the origin), or is squared length (which is cheaper to compute). Normalize() scales the vector so that its length becomes 1. The Cross and Dot methods allow you to perform two kinds of vector multiplication.

Comparison of Vector Operations

The **Vector3D** class now has a complete set of methods for performing vector arithmetic, eliminating the need work with the individual elements of the vectors. You may either save the result of the operation in the calling vector or save the result in a new vector. The following table compares the six methods.

Operation	Save result in place	Save result in a new vector
Addition	a.Add(b)	c = a.Plus(b)
Subtraction	a.Subtract(b)	c = a.Minus(b)
Scaling	a.MultiplyBy(f)	c = a.Times(f)

See Also

Object3D, Group3D, Quaternion classes; Rb3DSpace control.

VirtualVolume Class

Enables you to create and maintain a hierarchy of "virtual" files within one physical file. Basic reading and writing text and binary streams are supported.

Super Class

Object

Properties

Name	Type	Description
Root	FolderItem	Reference to the root directory of the virtual volume.

Notes

The FolderItem class has two methods and one property for working with virtual volumes. Use the CreateAsVirtualVolume and OpenAsVirtualVolume methods to create and open virtual volumes. The VirtualVolume property of the FolderItem class returns the **VirtualVolume** if the FolderItem is in a virtual volume.

Once you've created a **VirtualVolume**, you can get its Root and navigate to it using the same FolderItem methods that you would for real files. Virtual files are cross-platform compatible, and support basic reading and writing as text or binary files. The SaveAsPicture and SaveStyledEditField methods of the FolderItem class do not work for virtual volumes. Virtual volumes do not support resource forks.

Filenames can be up to 223 bytes long. Paths are not supported, but directories are (e.g., create a virtual directory with the CreateAsFolder method of the FolderItem class, and navigate to it with the Child or Item methods of this class). The virtual file system supports four-byte type codes (accessed via f.MacType), but does not support creator codes.

Example

The following example creates a **VirtualVolume**, gets its Root property, and writes a text file to the virtual volume.

```
Dim vv As VirtualVolume
Dim realFile As FolderItem
Dim virtFile As FolderItem
Dim outp As TextOutputStream

realFile = GetFolderItem("").Child("VV"))
vv = realFile.CreateVirtualVolume
If vv = Nil then
MsgBox "Unable to create virtual volume"
else
  virtFile = vv.Root.Child("Virtual File.txt")
  outp = virtFile.CreateTextFile
  outp.Write "Hello world!"
end if
```

To access an existing virtual volume, use the OpenAsVirtualVolume method instead of the CreateVirtualVolume method of the FolderItem class.

See Also

FolderItem class.

Volume Function

Returns a FolderItem that represents a mounted volume.

Syntax

result*=Volume*(VolumeNumber)

Part	Type	Description
result	FolderItem	The mounted volume whose number was passed.
VolumeNumber	Integer	The number of the volume you require a FolderItem for.

Notes

The **Volume** function returns a FolderItem that represents the mounted volume whose number was passed. Volume zero is the boot volume. This function can be used in conjunction with the VolumeCount function to loop through the mounted volumes.

Examples

This example places the names of all mounted volumes into a ListBox control:

```
Dim i,n as Integer
n= VolumeCount-1
For i=0 to n
 ListBox1.AddRow Volume(i).Name
Next
```

The following example returns a FolderItem for the "Mac OS 9" folder on the user's boot volume.

```
Dim f as FolderItem
f=Volume(0).Child("Mac OS 9")
```

See Also

VolumeCount function; FolderItem class.

VolumeCount Function

Returns the number of mounted volumes.

Syntax

***result*=VolumeCount**

Part	Type	Description
result	Integer	The number of mounted volumes.

Notes

The **VolumeCount** function returns the number of mounted volumes.

Examples

See the Volume function for an example.

See Also

Volume function.

While...Wend Statement

Repeatedly executes a series of statements while a specified condition is True.

Syntax

While *condition*

Statements

Wend

Condition is any valid Boolean expression.

Notes

If *Condition* is True, all statements are executed until the **Wend** statement is reached. If *Condition* is still True, the process is repeated. If *Condition* is False, then execution continues with the statement following the **Wend** statement.

While...Wend statements can be nested to any level. Each **Wend** statement goes with the previous **While** statement.

When a loop runs, it takes over the interface, preventing the user from interacting with menus and controls. Ordinarily, this is of no concern. If, however, the loop is very lengthy, you can move the code for the loop to a separate Thread, allowing it to execute in the background.

Example

This example uses the **While...Wend** statement to increment a variable.

```
Dim x As Integer
While x<100
  x=x+1
Wend
```

See Also

Do...Loop statement, For...Next statement.

Window Class

Any window.

Super Class

Object

Properties

Name	Type	Description
Backcolor	Color	The background color of the window.
Backdrop	Picture	A picture object that will be drawn in the window when the window opens.
Balloon Help	String	Balloon help text for the window.
Closebox	Boolean	If True, the window will include a Close box.
Composite	Boolean	If True, It allows controls to be used on top of other controls or a background picture without a rectangle surrounding the control. Does nothing for the following window types: Modal, Movable Modal, PlainBox, and ShadowedBox. Drawer and metal windows always have compositing on. Works on Mac OS X 10.2 and above only. See the Notes section for more information on the Composite property.

Name	Type	Description
Control	Array of Controls	Zero-based array of the controls in the window.
ControlCount	Integer	The number of controls in the window.
DockItem		(Mac OS X only). Enables you to manipulate the dock item associated with the window. The DockItem property has two methods, UpdateNow and ResetIcon. ResetIcon resets the icon to its original state (default appearance). Use the methods of the Graphics class to modify the appearance of the icon. (The Graphics property may be Nil; it is non-Nil only when an OS X window has been minimized to the dock.) Since a Mac OS X icon is intended to be scaled automatically, you should design it as a 128x128 pixel icon. Call the UpdateNow method to redraw the icon. You can also use the ClearRect property of the Graphics class to start over form a blank icon. Anything you can do with a Graphics object you are able to do with the Dock's Graphics object (like drawing in a picture, or using a Shape 2D). The DockItem property of the Application class enables you to control the DockItem for the whole application.
FloaterProcess	String	The Mac creator code for the application the window should display in front of. If this is empty, the window will float in front of all applications. The window type must be Global Floating Window for this to work.
Frame	Integer	The window type. 0-Document window 1-Movable Modal window 2-Modal Dialog box 3-Floating window 4-Plain box 5-Shadowed box 6-Rounded window 7-Global floating window 8-Sheet window (displayed as such on Mac OS X only) 9-Metal window (displayed as such on Mac OS X 10.2 and above only) 10-Drawer window (displayed as such on Mac OS X 10.2 and above only) See the section "Window Types" on page 44 of the *User's Guide* for illustrations and discussion.

Name	Type	Description
FullScreen	Boolean	When set to True, the window resizes itself to cover the entire screen. The values of the Left, Top, Width, and Height properties are adjusted to reflect the window's new position. They cannot be changed as long as FullScreen is True. If MenuBarVisible is set to True, the top of the window will be partially covered by the menu bar in Mac OS. In the case of multiple monitors, the window resizes to fit the monitor which contains the greatest portion of the window. FullScreen defaults to False.
GrowIcon	Boolean	If True, the window will be resizable.
HasBackColor	Boolean	If True, the background color of the window is set to the value of the Backcolor property.
Height	Integer	The height of the content region of the window.
Left	Integer	The distance (in pixels) between the left edge of the screen and the left edge of the content area of the window. The left frame of the window is taken into account.
LiveResize	Boolean	If True, the window is redrawn 'live' as it is resized by dragging. Setting LiveResize to True is recommended if you are using Drawer windows. Works on Mac OS X only.
MacProcID	Integer	Apple Window definition ID. Can be used to define window types not supported by the Frame property. Window types defined by MacProcID take effect only on the Macintosh platform. On Windows, the Window Type is controlled by the Frame property. See Notes section.
MacWindowPtr	Integer	Returns a pointer to the window. Can be used by toolbox programmers to control the window.
MaxHeight	Integer	The maximum height to which the window can be resized.
MaxWidth	Integer	The maximum width to which the window can be resized.
MenuBar	MenuItem	The menubar that is associated with the window. When the window is active, this menubar is displayed. If no menubar is assigned to the window, the application's global menubar is used. This is the menubar that is added to the project by default or is assigned using the Application class's MenuBar property. On Windows, every window uses a menubar just below its Title bar. It uses the menubar assigned to the window or the application's menubar if no menubar is assigned to the window. If it is an MDI application, the MDI window uses the Application's menubar. Individual document windows cannot have a menubar, but this is due to a limitation built into Windows. However, floating windows can have a menubar, and those windows will display the menubar you specify for them.

Name	Type	Description
MenuBarVisible	Boolean	When set to False, the menu bar is hidden whenever the window is the frontmost non-floating window. Set MenuBarVisible to False and FullScreen to True to allow the window to 'take over' the entire screen. The menu bar will be redisplayed if another window (whose MenuBarVisible property is set to True) becomes frontmost, or when the application is put into the background. MenuBarVisible defaults to True.
MinHeight	Integer	The minimum height to which the window can be resized.
MinWidth	Integer	The minimum width to which the window can be resized.
MouseCursor	MouseCursor	The cursor to be displayed while the mouse is within the window and the Application class's MouseCursor property is Nil. If the Application class MouseCursor is not Nil, non-Nil MouseCursors belonging to any Windows or Controls are ignored. If the Window's MouseCursor property is not Nil, the MouseCursor properties of any Controls are ignored. Possible values are: WatchCursor, IBeamCursor, and ArrowCursor. Cursors may also be assigned from Resource files that have been added to the project.
MouseX	Integer	The X coordinate of the mouse (pixels). Measured from the top-left corner of the window.
MouseY	Integer	The Y coordinate of the mouse (pixels). Measured from the top-left corner of the window.
Placement	Integer	The location where the window will be placed when it opens. 0 - Default 1 - Parent Window 2 - Main Screen 3 - Parent Window Screen 4 - Stagger For Drawer windows, the values have the following meaning: 0 - Top/Left 1 - Center 2 - Bottom/Right
Title	String	The window title.
Top	Integer	The distance (in pixels) between the top edge of the screen and the top edge of the content region of the window. The window's title bar is taken into account.
Visible	Boolean	If True, the window will be visible when it's opened.
Width	Integer	The width of the content region of the window. The left and right window frames are taken into account.
WinHWND	Integer	Returns a handle to the window (Win 32 only).
ZoomIcon	Boolean	If True, the window will include a zoom box.

Methods

Name	Parameters	Description
AcceptFileDrop	FileType as String	Permits documents of type *FileType* to be dropped on the window. *FileType* must be a file type you specified in the File Types dialog box. It works correctly even if you have multiple file types defined with the same File type code, and you don't specifically call AcceptFileDrop for the first such type.
AcceptMacDataDrop	Type as String	Permits data (of the type specified) to be dropped on the window. Type is a four-character resource code, e.g., 'snd ' or 'TEXT'
AcceptPictureDrop		Permits pictures to be dropped on the window.
AcceptTextDrop		Permits text to be dropped on the window.
Close		Closes the window. Once closed, a window cannot be refreshed or redrawn.
DrawInto	g as Graphics x as Integer y as Integer	Draws the contents of the window into the specified Graphics context.
Hide		Makes the window invisible.
Refresh		Repaints the entire contents of the window.
RefreshRect	X As Integer, Y As Integer, Width As Integer, Height As Integer	Repaints only the region of the window specified.
SetFocus		Removes the focus from the control that currently has the focus, leaving no control with the focus. Use the SetFocus method of the RectControl class to set the focus to a particular control in the window.
Show		Forces the window to immediately become visible rather than wait until the application is idle.
ShowModal		Causes the current method to stop executing until the dialog closes or becomes invisible.
ShowModalWithin	parentWindow as **Window**	Displays the window within the *parentWindow*. Used to display Sheet and Drawer windows within a specific parent.

Name	Parameters	Description
ShowWithin	parentWindow as **Window** [facing as Integer]	Displays the window within the *parentWindow*. Used to display Sheet and Drawer windows within a specific parent. For a Drawer window, the optional *facing* parameter indicates which side the drawer pops out of: -1 = default (determined by system -- usually left), 0 = top, 1 = bottom, 2 = right, 3 = left (same ordering as tab panels). If you omit *facing*, the default value will be used.
UpdateNow		Flushes the back buffer on Mac OS X windows. Because Mac OS X is double-buffered, drawing is accumulated before being shown on screen. You may want to flush the buffer manually after updating a progress bar, drawing the frame of an animation on a canvas, or other things of a similar nature. UpdateNow has no effect on Windows and Mac OS "classic".

Events

Name	Parameters	Description
Activate		The window is being activated.
CancelClose		The window is about to be closed by the user clicking the closebox or by the Quit method. Return True to prevent the window (and in the case of the Quit method, other open windows) from closing.
Close		The window is about to close.
Deactivate		The window is being deactivated.
DropObject	Obj as DragItem	The item represented by *Obj* has been dropped on the window.
EnableMenuItems		The user has clicked in the menu bar or pressed a keyboard shortcut assigned to one of the menu items.
KeyDown	Key as String	The Key parameter passed has been pressed and not handled by an object in the window.
MouseDown (function)	X as Integer, Y as Integer	The mouse button has been pressed inside the window at the x,y local coordinates passed. Return True if you are going to handle the mouseDown.
MouseDrag	X as Integer, Y as Integer	The mouse button was pressed inside the window and moved (dragged) at the location local to the window passed in to x,y. This event will not occur unless you return True in the MouseDown event.

Name	Parameters	Description
MouseEnter		The mouse has entered the window.
MouseExit		The mouse has left the window.
MouseMove	X as Integer, Y as Integer	The mouse has moved within the window to the x,y local coordinates passed.
MouseUp	X as Integer, Y as Integer	The mouse button has been released inside the window at the x,y local coordinates passed. This event will not occur unless you return True in the MouseDown event.
Moved		The window has been moved.
Open		The window is about to open.
Paint	g as Graphics	The window needs to be redrawn. The parameter passed gives you access to the Graphics class methods for drawing.
Resized		The window has been resized.
Resizing		The user is in the process of resizing the window. Resizing executes regardless of whether the LiveResize property is True in Carbon. Note: The Top, Left, Width and Height properties have already been adjusted to account for sizing at this point if LiveResize is on, but the controls haven't drawn themselves. You could, for example, use a Boolean property to indicate whether a Canvas control should redraw in its Paint event.

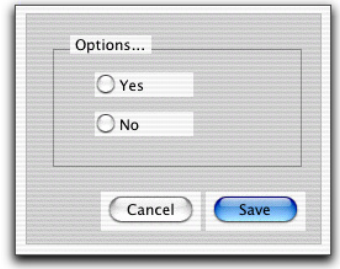
Notes

Window constructors are called after the window and its controls are created, but before the Open events are called.

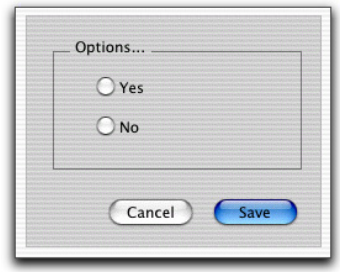
The Composite Property

The Composite property solves a display problem that occurs under Mac OS X. In certain cases, a control that is placed in a Metal or Drawer window or on top of another control shows its rectangular border as a white background unless the window's Composite property is turned on. The problem doesn't come up if the controls are intrinsically rectangular. For Drawer and Metal window types, the Composite property is turned on automatically. The problem can also occur in regular Document windows.

For example, controls that are placed on top of a Placard control in a Document window show this problem.



In this example, PushButtons, RadioButtons, and a GroupBox are inside a Placard control and the window's Composite property is off. Turning on the Composite property removes the unsightly white rectangles around each control.



Custom Window Types

If the MacProcID property is set to a non-zero value (which can be done in the IDE only), the window will be drawn according to this value. When you do so, the behaviour of the window will correspond to the value of the Frame property, but the appearance is controlled by the Window Definition provided by MacProcID.This feature allows you to define window types that are not defined directly by REALbasic (i.e, via the Frame property).

For a discussion of MacProcID values, see the Apple developer documentation "Window Definition IDs" at http://developer.apple.com/techpubs/macos8/HumanInterfaceToolbox/WindowManager/WindowMgr8Ref/WindowMgrRef.b.html.

The following table summarizes the possible values of MacProcID. Values of 1024 to 1087 require the Appearance Manager (Mac OS8 or above); other values are pre-Appearance Manager. On Windows, the value of the Frame property controls the window type regardless of the value of MacProcID.

Value	Description
1	Modal dialog box

Value	Description
2	Modeless dialog box
3	Modeless dialog box with shadow
4	Movable window with no size box
5	Movable dialog box with title bar
8	Movable window with zoom box and size box
12	Movable window with zoom box and no size box
16	Round corner window (adding 2, 4, or 6 to 16 changes the curvature of the corners)
1024	Movable window with no size box or zoom box
1025	Movable window with size box
1026	Movable window with vertical zoom box and no size box
1027	Movable window with vertical zoom box and size box
1028	Movable window with horizontal zoom box and no size box
1029	Movable window with horizontal zoom box and size box
1030	Movable window with full zoom box and no size box
1031	Movable window with full zoom box and size box
1040	Modeless dialog box
1041	Modeless dialog box with shadow
1042	Modal dialog box
1043	Movable modal dialog box
1044	Alert box
1045	Movable alert box
1046	Movable modal dialog box with size box
1057	Floating window with no size box or zoom box
1059	Floating window with size box
1061	Floating window with vertical zoom box
1063	Floating window with vertical zoom box and size box
1065	Floating window with horizontal zoom box
1067	Floating window with horizontal zoom box and size box
1069	Floating window with full zoom box
1071	Floating window with full zoom box and size box
1073	Floating window with side title bar
1075	Floating window with side title bar and size box
1077	Floating window with side title bar and vertical zoom box
1079	Floating window with side title bar, vertical zoom box, and size box
1081	Floating window with side title bar and horizontal zoom box
1083	Floating window with side title bar, horizontal zoom box, and size box

Value	Description
1085	Floating window with side title bar and full zoom box
1087	Floating window with side title bar, full zoom box, and size box
1985	Floating window with no size box or zoom box
1987	Floating window with size box
1989	Floating window with zoom box
1991	Floating window with zoom box and size box
1993	Floating window with side title bar and no size or zoom boxes
1995	Floating window with side title bar and size box
1997	Floating window with side title bar and zoom box
1999	Floating window with side title bar, size box, and zoom box.

Custom Cursors

The MouseCursor property controls the appearance of the pointer when it is over the window, provided the MouseCursor property of the Application class is Nil. If you also want the pointer to change to another shape when it is over a control within the window, you must either assign the new value to the Window's MouseCursor property or temporarily set the window's MouseCursor property to Nil and the control's MouseCursor property to the desired cursor. See the section on the MouseCursor class for an example.

Examples

This example sets the background color of the window to grey, provided the HasBackColor property is set to True.

```
backcolor=RGB(80,80,80)
```

This example increases the width of the window by 20 pixels.

```
width=width+20
```

This example changes the title of the window.

```
title="Document 1"
```

Drawinto Example

This example draws the contents of a window into a graphics object. For example, if you create a new project, add a second window, and add a Canvas control to the second window then in the Paint event of the first window, put:

```
Self.drawinto(dialog1.canvas1.graphics,0,0)
```

The contents of the first window will be draw into the canvas control on the second window.

Accessing Controls and Properties in Other Windows

When changing a window property from another window, there are two possible approaches you can take. If you have only one instance of the window you need to reference, you can use the window's object name as a reference. For example, if you had a window called window1 that had an EditField called EditField1 and you wanted to assign the value "Fred" to the text property of that EditField, you would using the following syntax:

```
window1.EditField1.text="Fred"
```

If you have multiple instances of the same window class open at the same time, then a reference to the target window must be included as in this example where a new window is opened and its window title changed. "anotherWindow" is a window class in the Project window.

```
Dim w As anotherWindow
w=New anotherWindow
w.title="Your Results"
```

Changing The Cursor

The MouseCursor property for the window controls the cursor that is displayed when the mouse is inside the window, providing the Application class's MouseCursor property is Nil. If the Application class's MouseCursor is not Nil, the MouseCursor property of the window is ignored. See the Application and Control classes for a detailed explanation.

Handling Drag and Drop

For an explanation of handling drag and drop, see the Control class and the DragItem class.

See Also

Window function.

Window Function

Returns a reference to an open window.

Syntax

result*=Window*(WindowNumber)

Part	Type	Description
result	Window	A reference to the window whose number was passed.
WindowNumber	Integer	The number of the window you want a reference to.

Notes

The **Window** function returns a reference to the window number passed. Window zero is the frontmost window. This function can be used in conjunction with the WindowCount function to loop through the open windows.

Example This example places the titles of all open windows into a ListBox:

```
Dim i as Integer
Dim WindowTitle as String
If WindowCount > 1 then
 For i=0 to WindowCount-1
  WindowTitle=Window(i).Title
  ListBox1.AddRow windowTitle
 Next
End If
```

See Also WindowCount function; Window class.

WindowCount Function

Returns the number of open windows.

Syntax ***result*=WindowCount**

Part	Type	Description
result	Integer	The number of open windows.

Notes The **WindowCount** function returns the number of open windows. This includes any windows that are invisible.

Examples See the Window function for an example.

See Also Window function.

You Can Only Inherit From a Class Error

Occurs only in RBScript. It occurs if you set a class's Super class to an interface instead of a real class.

You Can't Assign one Array to Another Array Error

You tried to assign one array to another array. To create an array that has the same elements of another array, you need to assign each value of one array to the same index

value of the other array. Use a For statement and loop through all the values of the array index.

Example

```
Dim a(5) as Integer
Dim b(5) as Integer
a=b
```

See Also

Dim statement.

You Can't Pass an Array to an External Function Error

You'll get this on the Declare statement if you define one of its parameters as an array.

See Also

Declare statement.

You Can't Pass an Expression as a Parameter that is Defined as ByRef Error

In a function call, you tried to pass an expression rather than a variable or property to a parameter that was declared as ByRef in the called method.

Example

The following method takes one parameter that was declared as ByRef:

```
Sub Squarit(ByRef c as Double)
 c=c*c
End Sub
```

The calling function is:

```
Dim a,b as Double
a=5
b=10
Squarit(a*b) //cannot use an expression here
```

See Also

ByRef operator.

You Can't Use Super Because this Class has no Super Class Error

You can't refer to the Super Class of a class that has no Super Class.

You Can't Use the New Operator with this Class Error

You tried to create an instance from an abstract class. You can only use the New operator to create a new instance of a control already in a window, a window added to the project with the File ▶ New Window command, or an existing menu item. The existing object acts as a template for the new instance.

You also cannot create an instance of a class that is used only as the base class for other classes, such as SocketCore.

Example

Don't do this:

```
Dim w as Window
w=New Window
```

Instead, use the New operator with an instance of the Window class created with the File ▶ New Window command.

```
Dim w as FindWindow
w=New FindWindow
```

See Also

New operator.

You Can't Use This Variable or Property Name Because it is a Reserved Word Error

Reserved words are not available as names of user-defined objects. If you used a reserved word as a variable name in a Dim statement or as a Property Name, this message will appear.

Examples

Using a reserved word, such as a data type (Integer, Double, Boolean, etc.) as the name of a property in the New Property dialog box.

You Cannot Implement a Nonexistent Event Error

Occurs only in RBScript. You attempted to use an event that is not defined in REALbasic.

See Also RBScript control.

You Cannot Return a Value Because this Method has No Defined Return Type Error

You can't use the Return statement in a method unless you define a data type for the value being returned in the function declaration.

Example

```
Sub myMethod(a as Integer, b as Integer)
 Return a*b
End Sub
```

To return a value, declare the data type of the returned value. In that case, the declaration statement in this example would be:

```
Function myMethod(a as Integer, b as Integer) as Integer
```

See Also Sub, Function statements.

You Cannot Use the New Operator with a Class Interface Error

You used the New operator with a class interface. You create new class interfaces using the File ▶ New Class Interface command.

Example The class interface Interface1 has already been added to the project.

```
Dim c as Interface1
c=New Interface1
```

You Have Used an Operator that is not Compatible with the Data Types Specified Error

You tried to perform an operation on data types that don't support the operation. For example, multiplying strings, dividing Booleans, etc.

Examples

```
Dim c as Integer
Dim d,e as String
Dim g,h as Boolean
d="Ted"
e="Alice"
c=d*e //error
c=g/h //error
```

You Must Indicate the Data Type of the Value to be Returned Error

You declared an external function using the Function keyword but did not specify the return type. This happens because the data type of the value returned was left off or because the programmer intended to declare a sub but used function instead.

See Also

Function, Sub statements.

You Must Use the Value Returned by this Function Error

You called a function but did not assign the result to a variable or property.

Note

If you want to call the function but ignore the result, use the Call statement.

Examples

Trying to assign the result to an object rather than a property of the object:

```
CheckBox1=True
```

Ignoring the result returned by a built-in function:

```
Atan2(1,0) //not assigned to anything
RGB(100,100,100) //incorrect
```

See Also

Function, Call statements.

APPENDIX A

Text Encoding Base and Variant Codes

The tables in this appendix give the values of the *Base* and *Variant* parameters that are used in several of the functions in the Text Encoding theme. Table 3 gives the codes for *Base* and Table 4 gives the codes for *Variant*.

Table 3: Text Encoding codes for Base

Base	Value
MacRoman	0
MacJapanese	1
MacChineseTrad	2
MacKorean	3
MacArabic	4
MacHebrew	5
MacGreek	6
MacCyrillic	7
MacDevanagari	9
MacGurmukhi	10
MacGujarati	11
MacOriya	12
MacBengali	13
MacTamil	14
MacTelugu	15
MacKannada	16
MacMalayalam	17
MacSinhalese	18
MacBurmese	19
MacKhmer	20
MacThai	21
MacLaotian	22
MacGeorgian	23
MacArmenian	24
MacChineseSimp	25
MacTibetan	26
MacMongolian	27
MacEthiopic	28
MacCentralEurRoman	29
MacVietnamese	30
MacExtArabic	31
MacSymbol	33
MacDingbats	34

Table 3: Text Encoding codes for Base (Continued)

Base	Value
MacTurkish	35
MacCroatian	36
MacIcelandic	37
MacRomanian	38
MacCeltic	39
MacGaelic	40
MacUnicode	&h7E
MacFarsi	&h8C
MacUkrainian	&h98
MacInuit	&hEC
MacVT100	&hFC
MacHFS	&hFF
UnicodeDefault	&h0100
UnicodeV1_1	&h0101
ISO10646_1933	&h0101
UnicodeV2_0	&h0103
Unicodev2_1	&h0103
ISOLatin1	&h0201
ISOLatin2	&h0202
ISOLatin3	&h0203
ISOLatin4	&h0204
ISOLatinCyrillic	&h0205
ISOLatinArabic	&h0206
ISOLatinGreek	&h0207
ISOLatinHebrew	&h0208
ISOLatin5	&h0209
DOSLatinUS	&h0400
DOSGreek	&h0405
DOSBalticRim	&h0405
DOSLatin1	&h0410
DOSGreek1	&h0411
DOSLatin2	&h0412
DOSCyrillic	&h0413
DOSTurkish	&h0414
DOCPortuguese	&h0415

Table 3: Text Encoding codes for Base (Continued)

Base	Value
DOSIslandic	&h0416
DOSHebrew	&h0417
DOSCanadianFrench	&h0418
DOSArabic	&h0419
DOSNordic	&h041A
DOSRussian	&h041B
DOSGreek2	&h041C
DOSThai	&h041D
DOSJapanese	&h0420
DOSChineseSimplif	&h0421
DOSKorean	&h0422
DOSChineseTrad	&h0423
WindowsLatin1	&h0500
WindowsANSI	&h0500
WindowsLatin2	&h0501
WindowsCryillic	&h0502
WindowsGreek	&h0503
WindowsLatin5	&h0504
WindowsHebrew	&h0505
WindowsArabic	&h0506
WindowsBalticRim	&h0507
WindowsVietnamese	&h0508
WindowsKoreanJohab	&h0510
US_ASCII	&h0600
JIS_X0201_76	&h0620
JIS_X0208_83	&h0621
JIS_X0208_90	&h0622
JIS_X0212_90	&h0623
JIS_C6226_78	&h0624
GB_2312_80	&h0630
GBK_95	&h0631
KSC_5601_87	&h0640
KSC_5601_92_Johab	&h0641
CNS_11643_92_P1	&h0651
CNS_11643_92_P2	&h0652

Table 3: Text Encoding codes for Base (Continued)

Base	Value
CNS_11643_92_P3	&h0653
ISO_2022_JP	&h0820
ISO_2022_JP_2	&h0821
ISO_2022_CN	&h0830
ISO_2022_CN_EXT	&h0831
ISO_2022_KR	&h0840
EUC_JP	&h0920
EUC_CN	&h0930
EUC_TW	&h0931
EUC_KR	&h0940
ShiftJIS	&h0A01
KOI8_R	&h0A02
Big5	&h0A03
MacRomanLatin1	&h0A04
HZ_GB_2312	&h0A05
NextStepLatin	&h0B01
EBCDIC_US	&h0C01
EBCDIC_CP037	&h0C02
MultiRun	&h0FFF
MacTradChinese	2
MacRSSymbol	8
MacSimpChinese	25
MacGeez	28
MacEastEurRoman	29
MacUninterp	32

Table 4: Text Encoding codes for Variant.

Variant	Value
Text Encoding Default Variant	0
Mac Roman Default Variant	0
Mac Roman Currency Sign Variant	1
Mac Roman Euro Sign Variant	2
Mac Icelandic Std Default Variant	0
Mac Icelandic TT Default Variant	1
Mac Icelandic Std Euro Sign Variant	4
Mac Icelandic TT Euro Sign Variant	5
Mac Croatian Default Variant	0
Mac Croatian Currency Sign Variant	1
Mac Croatian Euro Sign Variant	2
Mac Romanian Default Variant	0
Mac Romanian Currency Sign Variant	1
Mac Romanian Euro Sign Variant	2
Mac Japanese Standard Variant	0
Mac Japanese StdNoVerticals Variant	1
Mac Japanese Basic Variant	2
Mac JapanesePostScript Scrn Variant	3
Mac Japanese PostScript Print Variant	4
Mac Japanese VartAtKuPlusTen Variant	5
Mac Arabic Standard Variant	0
Mac Arabic TrueType Variant	1
Mac Arabic Thuluth Variant	2
Mac Arabic AlBayan Variant	3
Mac Farsi Standard Variant	0
Mac Farsi TrueType Variant	1
Mac Hebrew Standard Variant	0
Mac Hebrew FigureSpace Variant	1
Mac VT100 Default Variant	0
Mac VT100 CurrencySign Variant	1
Mac VT100 EuroSign Variant	2
Unicode No Subset	0
Unicode Canonical Decomp Variant	2

Table 4: Text Encoding codes for Variant. (Continued)

Variant	Value
Big5_Basic Variant	0
Big5_Standard Variant	1
Big5_ETen Variant	2
Unicode No Compatibility Variant	1
Unicode No Composed Variant	3
Unicode No Corporate Variant	4
Mac Roman Standard Variant	0
Mac Icelandic Standard Variant	0
Mac Icelandic TrueType Variant	1
Japanese Standard Variant	0
Japanese Std No Verticals Variant	1
Japanese Basic Variant	2
Japanese PostScript Scrn Variant	3
Japanese PostScript Print Variant	4
Japanses VertAtKuPlusTen Variant	5
Hebrew Standard Variant	0
Hebrew Figure Space Variant	1
Unicode Max Decomposed Variant	2
Japanese NoOneByteKanaOption	0020
Japanese Use ASCII Backslash Option	&h40

APPENDIX B

ASCII Codes

The tables in this appendix presents ASCII codes for both Macintosh and Windows. Table 5 presents the Decimal, Hex, and Octal values for standard ASCII (0 to 127); Table 6 presents the high ASCII character sets for the Mac and Windows platforms.

Table 5: Standard ASCII codes.

Decimal	Hex	Octal	Result	Decimal	Hex	Octal	Result
0	0	0	NUL	32	20	40	SP
1	1	1	SOH	33	21	41	!
2	2	2	STX	34	22	42	"
3	3	3	ETX	35	23	43	#
4	4	4	EOT	36	24	44	$
5	5	5	ENQ	37	25	45	%
6	6	6	ACK	38	26	46	&
7	7	7	BEL	39	27	47	'
8	8	10	BS	40	28	50	(
9	9	11	HT	41	29	51	)
10	A	12	LF	42	2A	52	*
11	B	13	VT	43	2B	53	+
12	C	14	FF	44	2C	54	,
13	D	15	CR	45	2D	55	-
14	E	16	SO	46	2E	56	.
15	F	17	SI	47	2F	57	/
16	10	20	DLE	48	30	60	0
17	11	21	DC1	49	31	61	1
18	12	22	DC2	50	32	62	2
19	13	23	DC3	51	33	63	3
20	14	24	DC4	52	34	64	4
21	15	25	NAK	53	35	65	5
22	16	26	SYN	54	36	66	6
23	17	27	ETB	55	37	67	7
24	18	30	CAN	56	38	70	8
25	19	31	EM	57	39	71	9
26	1A	32	SUB	58	3A	72	:
27	1B	33	ESC	59	3B	73	;
28	1C	34	FS	60	3C	74	<
29	1D	35	GS	61	3D	75	=
30	1E	36	RS	62	3E	76	>
31	1F	37	US	63	3F	77	?

Table 5: Standard ASCII codes. (Continued)

Decimal	Hex	Octal	Result	Decimal	Hex	Octal	Result
64	40	100	@	96	60	140	`
65	41	101	A	97	61	141	a
66	42	102	B	98	62	142	b
67	43	103	C	99	63	143	c
68	44	104	D	100	64	144	d
69	45	105	E	101	65	145	e
70	46	106	F	102	66	146	f
71	47	107	G	103	67	147	g
72	48	110	H	104	68	150	h
73	49	111	I	105	69	151	i
74	4A	112	J	106	6A	152	j
75	4B	113	K	107	6B	153	k
76	4C	114	L	108	6C	154	l
77	4D	115	M	109	6D	155	m
78	4E	116	N	110	6E	156	n
79	4F	117	O	111	6F	157	o
80	50	120	P	112	70	160	p
81	51	121	Q	113	71	161	q
82	52	122	R	114	72	162	r
83	53	123	S	115	73	163	s
84	54	124	T	116	74	164	t
85	55	125	U	117	75	165	u
86	56	126	V	118	76	166	v
87	57	127	W	119	77	167	w
88	58	130	X	120	78	170	x
89	59	131	Y	121	79	171	y
90	5A	132	Z	122	7A	172	z
91	5B	133	[	123	7B	173	{
92	5C	134	\	124	7C	174	|
93	5D	135	]	125	7D	175	}
94	5E	136	^	126	7E	176	~
95	5F	137	-	127	7F	177	DEL

Table 6: Extended ASCII Codes (Macintosh and Windows).

Decimal	Hex	Octal	Result (Mac)	Result (Windows)	Decimal	Hex	Octal	Result (Mac)	Result (Windows)
128	80	200	Ä		160	A0	240	†	
129	81	201	Å		161	A1	241	°	¡
130	82	202	Ç	‚	162	A2	242	¢	¢
131	83	203	É	ƒ	163	A3	243	£	£
132	84	204	Ñ	„	164	A4	244	§	
133	85	205	Ö	…	165	A5	245	•	¥
134	86	206	Ü	†	166	A6	246	¶	¦
135	87	207	á	‡	167	A7	247	ß	§
136	88	210	à	ˆ	168	A8	250	®	¨
137	89	211	â	‰	169	A9	251	©	©
138	8A	212	ä	Š	170	AA	252	™	ª
139	8B	213	ã	‹	171	AB	253	´	«
140	8C	214	å	Œ	172	AC	254	¨	¬
141	8D	215	ç		173	AD	255	≠	-
142	8E	216	é		174	AE	256	Æ	®
143	8F	217	è		175	AF	257	Ø	¯
144	90	220	ê		176	B0	260	∞	°
145	91	221	ë	‘	177	B1	261	±	±
146	92	222	í	’	178	B2	262	≤	²
147	93	223	ì	“	179	B3	263	≥	³
148	94	224	î	”	180	B4	264	¥	´
149	95	225	ï	•	181	B5	265	µ	µ
150	96	226	ñ	–	182	B6	266	∂	¶
151	97	227	ó	—	183	B7	267	∑	·
152	98	230	ò	˜	184	B8	270	∏	¸
153	99	231	ô	™	185	B9	271	π	¹
154	9A	232	ö	š	186	BA	272	∫	º
155	9B	233	õ	›	187	BB	273	ª	»
156	9C	234	ú	œ	188	BC	274	º	¼
157	9D	235	ù		189	BD	275	Ω	½
158	9E	236	û		190	BE	276	æ	¾
159	9F	237	ü	Ÿ	191	BF	277	ø	¿

Table 6: Extended ASCII Codes (Macintosh and Windows). (Continued)

Decimal	Hex	Octal	Result (Mac)	Result (Windows)	Decimal	Hex	Octal	Result (Mac)	Result (Windows)
192	C0	300	¿	À	224	E0	340	‡	à
193	C1	301	¡	Á	225	E1	341	·	á
194	C2	302	¬	Â	226	E2	342	‚	â
195	C3	303	√	Ã	227	E3	343	„	ã
196	C4	304	ƒ	Ä	228	E4	344	‰	å
197	C5	305	≈	Å	229	E5	345	Â	å
198	C6	306	∆	Æ	230	E6	346	Ê	æ
199	C7	307	«	Ç	231	E7	347	Á	ç
200	C8	310	»	È	232	E8	350	Ë	è
201	C9	311	…	É	233	E9	351	È	é
202	CA	312	(space)	Ê	234	EA	352	Í	ê
203	CB	313	À	Ë	235	EB	353	Î	ë
204	CC	314	Ã	Ì	236	EC	354	Ï	ì
205	CD	315	Õ	Í	237	ED	355	Ì	í
206	CE	316	Œ	Î	238	EE	356	Ó	î
207	CF	317	œ	Ï	239	EF	357	Ô	ï
208	D0	320	–	Ð	240	F0	360		ð
209	D1	321	—	Ñ	241	F1	361	Ò	ñ
210	D2	322	“	Ò	242	F2	362	Ú	ò
211	D3	323	”	Ó	243	F3	363	Û	ó
212	D4	324	‘	Ô	244	F4	364	Ù	ô
213	D5	325	’	Õ	245	F5	365	ı	õ
214	D6	326	÷	Ö	246	F6	366	ˆ	ö
215	D7	327	◊	×	247	F7	367	˜	÷
216	D8	330	ÿ	Ø	248	F8	370	¯	ø
217	D9	331	Ÿ	Ù	249	F9	371	˘	ù
218	DA	332	⁄	Ú	250	FA	372	˙	ú
219	DB	333		Û	251	FB	373	˚	û
220	DC	334	‹	Ü	252	FC	374	¸	ü
221	DD	335	›	Ý	253	FD	375	˝	ý
222	DE	336	fi	Þ	254	FE	376	˛	þ
223	DF	337	fl	ß	255	FF	377	ˇ	ÿ

Alphabetical Command List

Commands by Theme

3D Graphics

AppleEvents

Arrays

Boolean

Classes

Code Execution

Constants

Controls

Data Types

Database

Date and Time

Debugging

Email

Error Handling

Error Messages

Files

Fonts

Graphics

Internet

Keychains

Math

Menus

Monitors/Screens

Multithreading

Networking

Object Binding

Objects

OLE

Operators

OS

Printing

QuickTime

Registry Items

Regular Expressions

Resources

Runtime Errors

Sound

Strings

Text Encoding

Toolbars

Variables

Vector Graphics

Windows