SYBASE®

PowerScript® Reference

PocketBuilder™

2.5

DOCUMENT ID: DC00132-01-0250-01

LAST REVISED: December 2007

Contents

About This Book

Audience

This guide is for programmers building applications with PocketBuilder™.

How to use this book

This book describes syntax and usage information for the PowerScript® language, including variables, expressions, statements, events, and functions.

Related documents

PocketBuilder reference set This manual is part of the PocketBuilder reference set, which is based on PowerBuilder® documentation. The reference set also includes the following manuals:

- *Connection Reference* - Describes the database parameters and preferences you use to connect to a database in PocketBuilder.
- *DataWindow Reference* - Lists the DataWindow® functions and properties and includes the syntax for accessing properties and data in DataWindow objects.
- *Objects and Controls* - Describes the system-defined objects and their default properties, functions, and events.

PocketBuilder documentation set The PocketBuilder documentation set includes the following manuals:

- *Introduction to PocketBuilder* - Provides an overview of PocketBuilder features and the PocketBuilder development environment and a tutorial that leads the new user through the basic process of creating and deploying PocketBuilder applications.
- *Resource Guide* - Presents advanced programming techniques and information about connecting to and synchronizing with a database.
- *Users Guide* - Gives an overview of the PocketBuilder development environment and explains how to use the interface. Describes basic techniques for building the objects in a PocketBuilder application, including windows, menus, DataWindow objects, and user-defined objects. An appendix summarizes the differences between PocketBuilder and PowerBuilder.

Online Help Reference information for PowerScript properties, events, and functions is available in the online Help with annotations indicating which objects and methods are applicable to PowerBuilder.

SQL Anywhere® documentation PocketBuilder is tightly integrated with the SQL Anywhere database server and management system (formerly Adaptive Server Anywhere), including its UltraLite®, MobiLink™, and Sybase Central™ components. You can install these products from the PocketBuilder setup program. For an introduction to these products, see Chapter 1 in the *Introduction to PocketBuilder*. Documentation for SQL Anywhere is available on the iAnywhere Web site at http://www.ianywhere.com/developer/product_manuals/sqlanywhere/.

Other sources of information

Use the Sybase Getting Started CD, the SyBooks CD, and the Sybase Product Manuals Web site to learn more about your product:

- The Getting Started CD contains release bulletins and installation guides in PDF format, and may also contain other documents or updated information not included on the SyBooks CD. It is included with your software. To read or print documents on the Getting Started CD, you need Adobe Acrobat Reader, which you can download at no charge from the Adobe Web site using a link provided on the CD.
- The SyBooks CD contains product manuals and is included with your software. The Eclipse-based SyBooks browser allows you to access the manuals in an easy-to-use, HTML-based format.

  Some documentation may be provided in PDF format, which you can access through the PDF directory on the SyBooks CD. To read or print the PDF files, you need Adobe Acrobat Reader.

  Refer to the *SyBooks Installation Guide* on the Getting Started CD, or the *README.txt* file on the SyBooks CD for instructions on installing and starting SyBooks.
- The Sybase Product Manuals Web site is an online version of the SyBooks CD that you can access using a standard Web browser. In addition to product manuals, you will find links to EBFs/Maintenance, Technical Documents, Case Management, Solved Cases, newsgroups, and the Sybase Developer Network.

  To access the Sybase Product Manuals Web site, go to Product Manuals at http://www.sybase.com/support/manuals/.

Sybase EBFs and software maintenance

❖ Finding the latest information on EBFs and software maintenance

1. Point your Web browser to the Sybase Support Page at http://www.sybase.com/support.
2. Select EBFs/Maintenance. If prompted, enter your MySybase user name and password.
3. Select a product.
4. Specify a time frame and click Go. A list of EBF/Maintenance releases is displayed.

   Padlock icons indicate that you do not have download authorization for certain EBF/Maintenance releases because you are not registered as a Technical Support Contact. If you have not registered, but have valid information provided by your Sybase representative or through your support contract, click Edit Roles to add the "Technical Support Contact" role to your MySybase profile.
5. Click the Info icon to display the EBF/Maintenance report, or click the product description to download the software.

Conventions

The formatting conventions used in this manual are:

Formatting example	To indicate
Retrieve and Update	When used in descriptive text, this font indicates: • Command, function, and method names • Keywords such as true, false, and null • Datatypes such as integer and char • Database column names such as emp_id and f_name • User-defined objects such as dw_emp or w_main
variable or *file name*	When used in descriptive text and syntax descriptions, oblique font indicates: • Variables, such as *myCounter* • Parts of input text that must be substituted, such as *pklname*.`pkd` • File and path names

Formatting example	To indicate
File>Save	Menu names and menu items are displayed in plain text. The greater than symbol (>) shows you how to navigate menu selections. For example, File>Save indicates “select Save from the File menu.”
`dw_1.Update()`	Monospace font indicates: • Information that you enter in a dialog box or on a command line • Sample script fragments • Sample output fragments

If you need help

Each Sybase installation that has purchased a support contract has one or more designated people who are authorized to contact Sybase Technical Support. If you cannot resolve a problem using the manuals or online help, please have the designated person contact Sybase Technical Support or the Sybase subsidiary in your area.

PART 1

PowerScript Topics

CHAPTER 1

Language Basics

About this chapter

This chapter describes general elements and conventions of PowerScript.

Contents

Comments

Description

You can use comments to document your scripts and prevent statements within a script from executing. There are two methods.

Syntax

Double-slash method

Code // *Comment*

Slash-and-asterisk method

/* *Comment* */

Usage

The following table shows how to use each method.

Table 1-1: Methods for adding comments in scripts

Method	Marker	Can use to	Note
Double slash	//	Designate all text on the line to the right of the marker as a comment	*Cannot* extend to multiple lines
Slash and asterisk	/*...*/	Designate the text between the markers as a comment Nest comments	• Can extend over multiple lines (multiline comments do not require a continuation character) • Can be nested

Adding comment markers
In Script views and the Function painter, you can use the Comment Selection button (or select Edit>Comment Selection from the menu bar) to comment out the line containing the cursor or a selected group of lines.

For information about adding comments to objects and library entries, see the *Users Guide*.

Examples

Double-slash method

```
// This entire line is a comment.
// This entire line is another comment.
amt = qty * cost // Rest of the line is comment.

// The following statement was commented out so that it
// would not execute.
// SetNull(amt)
```

Slash-and-asterisk method

```
/* This is a single-line comment.   */

/* This comment starts here,
continues to this line,
and finally ends here. */

A = B + C  /*  This comment starts here.
/*  This is the start of a nested comment.
    The nested comment ends here.  */
The first comment ends here.  */ + D + E + F
```

Identifier names

Description

You use identifiers to name variables, labels, functions, windows, controls, menus, and anything else you refer to in scripts.

Syntax

Rules for identifiers:

- Must start with a letter or an _ (underscore)
- Cannot be reserved words (see "Reserved words" on page 9)
- Can have up to 40 characters but no spaces
- Are not case sensitive (PART, Part, and part are identical)
- Can include any combination of letters, numbers, and these special characters:

-	Dash
_	Underscore
$	Dollar sign
#	Number sign
%	Percent sign

Usage

By default, PocketBuilder allows you to use dashes in all identifiers, including in variable names in a script. However, this means that when you use the subtraction operator or the -- operator in a script, you must surround it with spaces. If you do not, PocketBuilder interprets the expression as an identifier name.

If you want to disallow dashes in variable names in scripts, you can change the setting of the Allow Dashes in Identifiers option in the script editor's property sheet. As a result, you do not have to surround the subtraction operator and the decrement assignment shortcut (--) with spaces.

Be careful
If you disallow dashes and have previously used dashes in variable names, you will get errors the next time you compile.

Examples

Valid identifiers

```
ABC_Code
Child-Id
FirstButton
response35
pay-before%deductions$
ORDER_DATE
```

```
Actual-$-amount
Part#
```

Invalid identifiers

```
2nd-quantity // Does not start with a letter
ABC Code     // Contains a space
Child'sId    // Contains invalid special character
```

Labels

Description

You can include labels in scripts for use with GOTO statements.

Syntax

Identifier :

Usage

A label can be any valid identifier. You can enter it on a line by itself above the statement or at the start of the line before the statement.

For information about the GOTO statement, see GOTO on page 129. For information about valid identifiers, see “Identifier names” on page 5.

Examples

On a line by itself above the statement

```
FindCity:
IF city=cityname[1] THEN ...
```

At the start of the line before the statement

```
FindCity: IF city=cityname[1] THEN ...
```

Special ASCII characters

Description

You can include special ASCII characters in strings. For example, you might want to include a tab in a string to ensure proper spacing or a bullet to indicate a list item. The tilde character (~) introduces special characters. The tab is one of the common ASCII characters that can be entered by typing a tilde followed by a single keystroke. The bullet must be entered by typing a tilde followed by the decimal, hexadecimal, or octal ASCII value that represents it.

Syntax

Follow the guidelines in the following table.

Table 1-2: Using special ASCII characters in strings

In this category	To specify this	Enter this	More information
Common ASCII characters	Newline	~n	
	Tab	~t	
	Vertical tab	~v	
	Carriage return	~r	
	Form feed	~f	
	Backspace	~b	
	Double quote	~"	
	Single quote	~'	
	Tilde	~~	
Any ASCII character	Decimal	~###	### = a 3-digit number from 000 to 255
	Hexadecimal	~h##	## = a 2-digit hexadecimal number from 01 to FF
	Octal	~o###	### = a 3-digit octal number from 000 to 377

Examples

Entering ASCII characters Here is how to use special characters in strings:

String	Description
"dog~n"	A string containing the word dog followed by a newline character
"dog~tcat~ttiger"	A string containing the word *dog*, a tab character, the word *cat*, another tab character, and the word *tiger*

Using decimal, hexadecimal, and octal values Here is how to indicate a bullet (•) in a string by using the decimal, hexadecimal, and octal ASCII values:

Value	Description
"~249"	The ASCII character with decimal value 249
"~hF9"	The ASCII character with hexadecimal value F9
"~o371"	The ASCII character with octal value 371

NULL values

Description

Null means *undefined* or *unknown*. It is not the same as an empty string or zero or a date of 0000-00-00. For example, null is neither 0 nor not 0.

Typically, you work with null values only with respect to database values.

Usage

Initial values for variables Although PocketBuilder supports null values for all variable datatypes, it does *not* initialize variables to null. Instead, when a variable is not set to a specific value when it is declared, PocketBuilder sets it to the default initial value for the datatype—for example, zero for a numeric value, false for boolean, and the empty string ("") for a string.

Null variables A variable can become null if one of the following occurs:

- A null value is read into it from the database. If your database supports null, and a SQL INSERT or UPDATE statement sends a null to the database, it is written to the database as null and can be read into a variable by a SELECT or FETCH statement.

  Null in a variable
  When a null value is read into a variable, the variable remains null unless it is changed in a script.

- The SetNull function is used in a script to set the variable explicitly to null. For example:

```
string city      // city is an empty string.
SetNull(city)    // city is set to NULL.
```

Nulls in functions and expressions Most functions that have a null value for *any* argument return null. Any expression that has a variable with a null value results in null.

A boolean expression that is null is considered undefined and therefore false.

Testing for null To test whether a variable or expression is null, use the IsNull function. You *cannot* use an equal sign (=) to test for null.

Valid This statement shows the correct way to test for null:

```
IF IsNull(a) THEN      ...
```

Invalid This statement shows the incorrect way to test for null:

```
IF a = NULL THEN       ...
```

Examples

Example 1 None of the following statements make the computer beep (the variable *nbr* is set to null, so each statement evaluates to false):

```
int     Nbr
// Set Nbr to NULL.
SetNull(Nbr)
IF Nbr = 1 THEN Beep(1)
IF Nbr <> 1 THEN Beep(1)
IF NOT (Nbr = 1) THEN Beep(1)
```

Example 2 In this IF...THEN statement, the boolean expression evaluates to false, so the ELSE is executed:

```
int     a
SetNull(a)
IF a = 1 THEN
     MessageBox("Value", "a = 1")
ELSE
     MessageBox("Value", "a = NULL")
END IF
```

Example 3 This example is a more useful application of a null boolean expression than Example 2. It displays a message if no control has focus. When no control has focus, GetFocus returns a null object reference, the boolean expression evaluates to false, and the ELSE is executed:

```
IF GetFocus( ) THEN
     . . .  // Some processing
ELSE
     MessageBox("Important", "Specify an option!")
END IF
```

Reserved words

The words PocketBuilder uses internally are called reserved words and *cannot be used as identifiers*. If you use a reserved word as an identifier, you get a compiler warning. Reserved words that are marked with an asterisk (*) can be used as function names.

Table 1-3: PowerScript reserved words

alias	event	not	static
and	execute	of	step
autoinstantiate	exit	on	subroutine
call	external	open*	super
case	false	or	system
catch	fetch	parent	systemread
choose	finally	post*	systemwrite
close*	first	prepare	then
commit	for	prior	this
connect	forward	private	throw
constant	from	privateread	throws
continue	function	privatewrite	to
create*	global	procedure	trigger
cursor	goto	protected	true
declare	halt	protectedread	try
delete	if	protectedwrite	type
describe*	immediate	prototypes	until
descriptor	indirect	public	update*
destroy	insert	readonly	updateblob
disconnect	into	ref	using
do	intrinsic	return	variables
dynamic	is	rollback	while
else	last	rpcfunc	with
elseif	library	select	within
end	loop	selectblob	_debug
enumerated	next	shared	

The PocketBuilder system class also includes private variables that you cannot use as identifiers. If you use a private variable as an identifier, you get an informational message and should rename your identifier.

Pronouns

Description

PowerScript has pronouns that allow you to make a general reference to an object or control. When you use a pronoun, the reference remains correct even if the name of the object or control changes.

Usage

You can use pronouns in function and event scripts wherever you would use an object's name. For example, you can use a pronoun to:

- Cause an event in an object or control

- Manipulate or change an object or control
- Obtain or change the setting of a property

The following table lists the PowerScript pronouns and summarizes their use.

Table 1-4: PowerScript pronouns

This pronoun	In a script for a	Refers to the
This	Window, custom user object, menu, application object, or control	Object or control itself
Parent	Control in a window	Window containing the control
	Control in a custom user object	Custom user object containing the control
	Menu	Item in the menu on the level above the current menu
Super	Descendent object or control	Parent
	Descendent window or user object	Immediate ancestor of the window or user object
	Control in a descendent window or user object	Immediate ancestor of the control's parent window or user object

ParentWindow property You can use the ParentWindow property of the Menu object like a pronoun in Menu scripts. It identifies the window that the menu is associated with when your program is running. For more information, see the *Users Guide*.

The rest of this section describes the individual pronouns in detail.

Parent pronoun

Description

Parent in a PocketBuilder script refers to the object that contains the current object.

Usage

You can use the pronoun Parent in scripts for:

- Controls in windows
- Custom user objects
- Menus

Where you use Parent determines what it references:

Window controls When you use Parent in a script for a control (such as a CommandButton), Parent refers to the window that contains the control.

User object controls When you use Parent in a script for a control in a custom user object, Parent refers to the user object.

Menus When you use Parent in a menu script, Parent refers to the menu item on the level above the menu the script is for.

Examples

Window controls If you include this statement in the script for the Clicked event in a CommandButton within a window, clicking the button closes the window containing the button:

```
Close(Parent)
```

If you include this statement in the script for the CommandButton, clicking the button displays a horizontal scroll bar within the window (sets the HScrollBar property of the window to true):

```
Parent.HScrollBar = TRUE
```

User object controls If you include this statement in a script for the Clicked event for a CheckBox in a user object, clicking the check box hides the user object:

```
Parent.Hide( )
```

If you include this statement in the script for the CheckBox, clicking the check box disables the user object (sets the Enabled property of the user object to false):

```
Parent.Enabled = FALSE
```

Menus If you include this statement in the script for the Clicked event in the menu item Select All under the menu item Select, clicking Select All disables the menu item Select:

```
Parent.Disable( )
```

If you include this statement in the script for the Clicked event in the menu item Select All, clicking Select All checks the menu item Select:

```
Parent.Checked = TRUE
```

This pronoun

Description

The pronoun This in a PocketBuilder script refers to the window, user object, menu, application object, or control that owns the current script.

Usage

Why include This Using This allows you to make ownership explicit. The following statement refers to the current object's X property:

```
This.X = This.X + 50
```

When optional but helpful In the script for an object or control, you can refer to the properties of the object or control without qualification, but it is good programming practice to include This to make the script clear and easy to read.

When required There are some circumstances when you *must* use This. When a global or local variable has the same name as an instance variable, PocketBuilder finds the global or local variable first. Qualifying the variable with This allows you to refer to the instance variable instead of the global variable.

EAServer restriction
You cannot use This to pass arguments in EAServer components.

Examples

Example 1 This statement in a script for a menu places a check mark next to the menu selection:

```
This.Check( )
```

Example 2 In this function call, This passes a reference to the object containing the script:

```
ReCalc(This)
```

Example 3 If you omit This, "x" in the following statement refers to a local variable x if there is one defined (the script adds 50 to the variable x, not to the X property of the control). It refers to the object's X property if there is no local variable:

```
x = x + 50
```

Example 4 Use This to ensure that you refer to the property. For example, in the following statement in the script for the Clicked event for a CommandButton, clicking the button changes the horizontal position of the button (changes the button's X property):

```
This.x = This.x + 50
```

Super pronoun

Description

When you write a PocketBuilder script for a descendant object or control, you can call scripts written for any ancestor. You can directly name the ancestor in the call, or you can use the reserved word Super to refer to the immediate ancestor.

Usage

Whether to use Super If you are calling an ancestor function, you only need to use Super if the descendant has a function with the same name and the same arguments as the ancestor function. Otherwise, you would simply call the function with no qualifiers.

Restrictions for Super You cannot use Super to call scripts associated with controls in the ancestor window. You can only use Super in an event or function associated with a direct descendant of the ancestor whose function is being called. Otherwise, the compiler returns a syntax error.

To call scripts associated with controls, use the CALL statement.

See the discussion of CALL on page 116.

Examples

Example 1 This example calls the ancestor function wf_myfunc (presumably the descendant also has a function called wf_myfunc):

```
Super::wf_myfunc(myarg1, myarg2)
```

This example must be part of a script or function in the descendent window, not one of the window's controls. For example, if it is in the Clicked event of a button on the descendent window, you get a syntax error when the script is compiled.

Supplying arguments
Be certain to supply the correct number of arguments for the ancestor function.

Example 2 This example in a CommandButton script calls the Clicked script for the CommandButton in the immediate ancestor window or user object:

```
Super::EVENT Clicked()
```

Statement continuation

Description

Although you typically put one statement on each line, you occasionally need to continue a statement to more than one line. The statement continuation character is the ampersand (&). (For the use of the ampersand character in accelerator keys, see the *Users Guide*.)

Syntax

```
Start of statement &
    more statement &
    end of statement
```

The ampersand must be the last nonwhite character on the line or the compiler considers it part of the statement.

For information about white space, see “White space” on page 16.

Usage

You do not use a continuation character for:

- **Continuing comments** *Do not* use a continuation character to continue a comment. The continuation character is considered part of the comment and is ignored by the compiler.
- **Continuing SQL statements** You *do not* need a continuation character to continue a SQL statement. In PocketBuilder, SQL statements always end with a semicolon (;), and the compiler considers everything from the start of a SQL statement to a semicolon to be part of the SQL statement. A continuation character in a SQL statement is considered part of the statement and usually causes an error.

Examples

Continuing a quoted string

One way Place an ampersand in the middle of the string and continue the string on the next line:

```
IF Employee_District = "Eastern United States and&
Eastern Canada" THEN ...
```

Note that any white space (such as tabs and spaces) before the ampersand and at the beginning of the continued line is part of the string.

A problem The following statement uses only the ampersand to continue the quoted string in the IF...THEN statement to another line; for readability, a tab has been added to indent the second line. The compiler includes the tab in the string, which might result in an error:

```
IF Employee_District = "Eastern United States and&
      Eastern Canada" THEN ...
```

A better way A better way to continue a quoted string is to enter a quotation mark before the continuation character (`'&` or `"&`, depending on whether the string is delimited by single or double quotation marks) at the end of the first line of the string and a plus sign and a quotation mark (`+'` or `+"`) at the start of the next line. This way, you do not inadvertently include unwanted characters (such as tabs or spaces) in the string literal:

```
IF Employee_District = "Eastern United States and "&
     +" Eastern Canada" THEN ...
```

The examples in the PocketBuilder documentation use this method to continue quoted strings.

Continuing a variable name *Do not* split a line by inserting the continuation character within a variable name. This causes an error and the statement fails, because the continuation character splits the variable name “Quantity”:

```
Total-Cost = Price * Quan&
     tity + (Tax + Shipping)
```

Statement separation

Description

Although you typically put one statement on each line, you occasionally want to combine multiple statements on a single line. The statement separation character is the semicolon (;).

Syntax

Statement1; *statement2*

Examples

The following line contains three short statements:

```
A = B + C;  D = E + F;  Count = Count + 1
```

White space

Description

Blanks, tabs, form feeds, and comments are forms of white space. The compiler treats white space as a delimiter and does not consider the number of white space characters.

Usage

White space in string literals The number of white space characters is preserved when they are part of a string literal (enclosed in single or double quotation marks).

Dashes in identifiers Unless you have prohibited the use of dashes in identifiers (see “Identifier names” on page 5), you must surround a dash used as a minus sign with spaces. Otherwise, PocketBuilder considers the dash as part of a variable name:

```
Order - Balance  // Subtracts Balance from Order
Order-Balance    // A variable named Order-Balance
```

Examples

Example 1 Here the spaces and the comment are white space, so the compiler ignores them:

```
A + B /*Adjustment factor */+C
```

Example 2 Here the spaces are within a string literal, so the compiler does not ignore them:

```
"The value of A + B is:"
```

CHAPTER 2

Datatypes

About this chapter

This chapter describes the PowerScript datatypes.

Contents

Standard datatypes

The datatypes

The standard datatypes in PocketBuilder are the familiar datatypes that are used in many programming languages, including char, integer, decimal, long, and string. In PowerScript, you use these datatypes to declare variables or arrays.

These are the standard PowerScript datatypes, followed by a description of each:

Blob	LongLong
Boolean	Long
Char or character	Real
Date	String
DateTime	Time
Decimal or Dec	UnsignedInteger, UnsignedInt, or UInt
Double	UnsignedLong or ULong
Integer or Int	

Blob

Binary large object. Used to store an unbounded amount of data (for example, generic binary, image, or large text such as a word-processing document).

Boolean

Contains TRUE or FALSE.

Char or character

A single ASCII character.

If you have character-based data that you will want to parse in an application, you might want to define it as an array of type char. Parsing a char array is easier and faster than parsing strings. If you will be passing character-based data to external functions, you might want to use char arrays instead of strings.

For more information about passing character-based data to external functions, see the *Resource Guide*. For information about datatype conversion when assigning strings to chars and vice versa, see "String and char datatypes in PocketBuilder" on page 72.

Using literals To assign a literal value, enclose the character in either single or double quotation marks. For example:

```
char c
c = 'T'
c = "T"
```

Date

The date, including the full year (1000 to 3000), the number of the month (01 to 12), and the day (01 to 31).

Using literals To assign a literal value, separate the year, month, and day with hyphens. For example:

```
1992-12-25  // December 25, 1992
1995-02-06  // February 6, 1995
```

DateTime

The date and time in a single datatype, used only for reading and writing DateTime values from and to a database. To convert DateTime values to datatypes that you can use in PocketBuilder, use:

- The Date(*datetime*) function to convert a DateTime value to a PocketBuilder date value after reading from a database
- The Time(*datetime*) function to convert a DateTime value to a PocketBuilder time value after reading from a database
- The DateTime (*date*, *time*) function to convert a date and (optional) time to a DateTime before writing to a DateTime column in a database.

Decimal or Dec

Signed decimal numbers with up to 18 digits. You can place the decimal point anywhere within the 18 digits—for example, 123.456, 0.000000000000000001 or 12345678901234.5678.

Using literals To assign a literal value, use any number with a decimal point and no exponent. The plus sign is optional (95 and +95 are the same). For numbers between zero and one, the zero to the left of the decimal point is optional (for example, 0.1 and .1 are the same). For whole numbers, zeros to the right of the decimal point are optional (32.00, 32.0, and 32. are all the same). For example:

```
12.34    0.005     14.0    -6500     +3.5555
```

Double	A signed floating-point number with 15 digits of precision and a range from 2.2250738585073E-308 to 1.79769313486231E+308.
Integer or Int	16-bit signed integers, from -32768 to +32767.

Using literals To assign a literal value, use any whole number (positive, negative, or zero). The leading plus sign is optional (18 and +18 are the same). For example:

```
1        123       1200       +55       -32
```

Long — 32-bit signed integers, from -2147483648 to +2147483647.

Using literals Use literals as for integers, but longer numbers are permitted.

LongLong — 64-bit signed integers, from -9223372036854775808 to 9223372036854775807.

Using literals Use literals as for integers, but longer numbers are permitted.

Real — A signed floating-point number with six digits of precision and a range from 1.175495E-38 to 3.402822E+38.

Using literals To assign a literal value, use a decimal value, followed by E, followed by an integer; no spaces are allowed. The decimal number before the E follows all the conventions specified above for decimal literals. The leading plus sign in the exponent (the integer following the E) is optional (3E5 and 3E+5 are the same). For example:

```
2E4        2.5E78   +6.02E3    -4.1E-2
-7.45E16   7.7E+8   3.2E-45
```

String — Any ASCII character with variable length (0 to 2147483647).

Most of the character-based data in your application, such as names, addresses, and so on, will be defined as strings. PowerScript provides many functions that you can use to manipulate strings, such as a function to convert characters in a string to uppercase and functions to remove leading and trailing blanks.

For more information about passing character-based data to external functions, see the *Resource Guide*. For information about datatype conversion when assigning strings to chars and vice versa, see “String and char datatypes in PocketBuilder” on page 72.

Using literals To assign a literal value, enclose as many as 1024 characters in either single or double quotes, including a string of zero length or an empty string. For example:

```
string s1
s1 = 'This is a string'
s1 = "This is a string"
```

You can embed a quotation mark in a string literal if you enclose the literal with the other quotation mark. For example, the following statements result in the string `Here's a string`:

```
string s1
s1 = "Here's a string."
```

You can also use a tilde (~) to embed a quotation mark in a string literal. For example:

```
string s1 = 'He said, "It~'s good!"'
```

Complex nesting When you nest a string within a string that is nested in another string, you can use tildes to tell the parser how to interpret the quotation marks. Each pass through the parser strips away the outermost quotes and interprets the character after each tilde as a literal. Two tildes become one tilde, and tilde-quote becomes the quote alone.

Example 1 This string has two levels of nesting:

```
"He said ~"she said ~~~"Hi ~~~" ~" "
```

The first pass results in:

```
He said "she said ~"Hi ~" "
```

The second pass results in:

```
she said "Hi"
```

The third pass results in:

```
Hi
```

Example 2 A more probable example is a string for the Modify function that sets a DataWindow property. The argument string often requires complex quotation marks (because you must specify one or more levels of nested strings). To understand the quotation marks, consider how PocketBuilder will parse the string. The following string is a possible argument for the Modify function; it mixes single and double quotes to reduce the number of tildes:

```
"bitmap_1.Invert='0~tIf(empstatus=~~'A~~',0,1)'"
```

The double quotes tell PocketBuilder to interpret the argument as a string. It contains the expression being assigned to the Invert property, which is also a string, so it must be quoted. The expression itself includes a nested string, the quoted A. First, PocketBuilder evaluates the argument for Modify and assigns the single-quoted string to the Invert property. In this pass through the string, it converts two tildes to one. The string assigned to Invert becomes:

```
'0[tab]If(empstatus=~'A~',0,1)'
```

Finally, PocketBuilder evaluates the property's expression, converting tilde-quote to quote, and sets the bitmap's colors accordingly.

Example 3 There are many ways to specify quotation marks for a particular set of nested strings. The following expressions for the Modify function all have the same end result:

```
"emp.Color = ~"0~tIf(stat=~~~"a~~~",255,16711680)~""
"emp.Color = ~"0~tIf(stat=~~'a~~',255,16711680)~""
"emp.Color = '0~tIf(stat=~~'a~~',255,16711680)'"
"emp.Color = ~"0~tIf(stat='a',255,16711680)~""
```

Rules for quotation marks and tildes When nesting quoted strings, the following rules of thumb might help:

- A tilde tells the parser that the next character should be taken as a literal, not a string terminator
- Pairs of single quotes (') can be used in place of pairs of tilde double quotes (~")
- Pairs of tilde tilde single quotes (~~') can be used in place of pairs of triple tilde double quotes (~~~")

Time	The time in 24-hour format, including the hour (00 to 23), minute (00 to 59), second (00 to 59), and fraction of second (up to six digits), with a range from 00:00:00 to 23:59:59:999999.

Using literals The time in 24-hour format, including the hour (00 to 23), minute (00 to 59), second (00 to 59), and fraction of second (up to six digits), with a range from 00:00:00 to 23:59:59.999999. You separate parts of the time with colons—except for fractional sections, which should be separated by a decimal point. For example:

```
21:09:15    // 15 seconds after 9:09 pm
06:00:00    // Exactly 6 am
10:29:59    // 1 second before 10:30 am
10:29:59.9  // 1/10 sec before 10:30 am
```

UnsignedInteger, UnsignedInt, or UInt	16-bit unsigned integers, from 0 to 65535.
UnsignedLong or ULong	32-bit unsigned integers, from 0 to 4294967295.

The Any datatype

General information

PocketBuilder also supports the Any datatype, which can hold any kind of value, including standard datatypes, objects, structures, and arrays. A variable whose type is Any is a chameleon datatype—it takes the datatype of the value assigned to it.

Do not use Any in EAServer component definition
The Any datatype is specific to PowerScript and is not supported in the IDL of an EAServer component. CORBA has a datatype called Any that can assume any legal IDL type at runtime, but it is not semantically equivalent to the PocketBuilder Any type. You must exclude the PocketBuilder Any datatype from the component interface definition, but you can use it within the component.

Declarations and assignments

You declare Any variables just as you do any other variable. You can also declare an array of Any variables, where each element of the array can have a different datatype.

You assign data to Any variables with standard assignment statements. You can assign an array to a simple Any variable.

After you assign a value to an Any variable, you can test the variable with the ClassName function and find out the actual datatype:

```
any la_spreadsheetdata
la_spreadsheetdata = ole_1.Object.cells(1,1).value
CHOOSE CASE ClassName(la_spreadsheetdata)
    CASE "integer"
        ...
    CASE "string"
        ...
END CHOOSE
```

These rules apply to Any assignments:

- You can assign anything into an Any variable.
- You must know the content of an Any variable to make assignments from the Any variable to a compatible datatype.

Restrictions

If the value of a simple Any variable is an array, you cannot access the elements of the array until you assign the value to an array variable of the appropriate datatype. This restriction does not apply to the opposite case of an array of Any variables—you can access each Any variable in the array.

If the value of an Any variable is a structure, you cannot use dot notation to access the elements of the structure until you assign the value to a structure of the appropriate datatype.

After a value has been assigned to an Any variable, it cannot be converted back to a generic Any variable without a datatype. Even if you set it to NULL, it retains the datatype of the assigned value until you assign another value.

Operations and expressions

You can perform operations on Any variables as long as the datatype of the data in the Any variable is appropriate to the operator. If the datatype is not appropriate to the operator, an execution error occurs.

For example, if instance variables *ia_1* and *ia_2* contain numeric data, this statement is valid:

```
any la_3
la_3 = ia_1 - ia_2
```

If *ia_1* and *ia_2* contain strings, you can use the concatenation operator:

```
any la_3
la_3 = ia_1 + ia_2
```

However, if *ia_1* contained a number and *ia_2* contained a string, you would get an execution error.

Datatype conversion functions PowerScript datatype conversion functions accept Any variables as arguments. When you call the function, the Any variable must contain data that can be converted to the specified type.

For example, if *ia_any* contains a string, you can assign it to a string variable:

```
ls_string = ia_any
```

If *ia_any* contains a number that you want to convert to a string, you can call the String function:

```
ls_string = String(ia_any)
```

Other functions If a function's prototype does not allow Any as a datatype for an argument, you cannot use an Any variable without a conversion function, even if it contains a value of the correct datatype. When you compile the script, you get compiler errors such as `Unknown function` or `Function not found`.

For example, the argument for the Len function refers to a string column in a DataWindow, but the expression itself has a type of Any:

```
IF Len(dw_notes.Object.Notes[1]) > 0 THEN  // Invalid
```

This works because the string value of the Any expression is explicitly converted to a string:

```
IF Len(String(dw_notes.Object.Notes[1])) > 0 THEN
```

Expressions whose datatype is Any Expressions that access data whose type is unknown when the script is compiled have a datatype of Any. These expressions include expressions or functions that access data in an OLE object or a DataWindow object:

```
myoleobject.application.cells(1,1).value
dw_1.Object.Data[1,1]
dw_1.Object.Data.empid[99]
```

The objects these expressions point to can change so that the type of data being accessed also changes.

Expressions that refer to DataWindow data can return arrays and structures and arrays of structures as Any variables. For best performance, assign the DataWindow expression to the appropriate array or structure without using an intermediate Any variable.

Overusing the Any datatype

Do not use Any variables as a substitute for selecting the correct datatype in your scripts. There are two reasons for this:

- **At execution time, using Any variables is slow** PocketBuilder must do much more processing to determine datatypes before it can make an assignment or perform an operation involving Any variables. In particular, an operation performed many times in a loop will suffer greatly if you use Any variables instead of variables of the appropriate type.
- **At compile time, using Any variables removes a layer of error checking from your programming** The PocketBuilder compiler makes sure datatypes are correct before code gets executed. With Any variables, errors that can be caught by the compiler are not found until the code is run.

System object datatypes

Objects as datatypes

System object datatypes are specific to PowerScript. You view a list of all the system objects by selecting the System tab in the Browser.

In building PocketBuilder applications, you manipulate objects such as windows, menus, CommandButtons, ListBoxes, and graphs. Internally, PocketBuilder defines each of these kinds of objects as a datatype. Usually you do not need to concern yourself with these objects as datatypes—you simply define the objects in a PocketBuilder painter and use them.

However, sometimes you need to understand how PocketBuilder maintains its system objects in a hierarchy of datatypes. For example, when you need to define instances of a window, you define variables whose datatype is window. When you need to create an instance of a menu to pop up in a window, you define a variable whose datatype is menu.

PocketBuilder maintains its system objects in a class hierarchy. Each type of object is a class. The classes form an inheritance hierarchy of ancestors and descendants.

Examples

All the classes shown in the Browser are actually datatypes that you can use in your applications. You can define variables whose type is any class.

For example, the following code defines window and menu variables:

```
window mywin
menu mymenu
```

If you have a series of buttons in a window and need to keep track of one of them (such as the last one clicked), you can declare a variable of type CommandButton and assign it the appropriate button in the window:

```
// Instance variable in a window
commandbutton LastClicked
// In Clicked event for a button in the window.
// Indicates that the button was the last one
// clicked by the user.
LastClicked = This
```

Because it is a CommandButton, the LastClicked variable has all the properties of a CommandButton. After the last assignment above, LastClicked's properties have the same values as the most recently clicked button in the window.

To learn more about working with instances of objects through datatypes, see "About objects" on page 74.

Enumerated datatypes

About enumerated datatypes

Like the system object datatypes, enumerated datatypes are specific to PowerScript. Enumerated datatypes are used in two ways:

- As arguments in functions
- To specify the properties of an object or control

You can list all the enumerated datatypes and their values by selecting the Enumerated tab in the Browser.

You cannot create your own enumerated datatypes. As an alternative, you can declare a set of constant variables and assign them initial values. See "Declaring constants" on page 44.

A variable of one of the enumerated datatypes can be assigned a fixed set of values. Values of enumerated datatypes always end with an exclamation point (!). For example, the enumerated datatype Alignment, which specifies the alignment of text, can be assigned one of the following three values: Center!, Left!, and Right!:

```
mle_edit.Alignment=Right!
```

Incorrect syntax
Do not enclose an enumerated datatype value in quotation marks. If you do, you receive a compiler error.

Advantages of enumerated types

Enumerated datatypes have an advantage over standard datatypes. When an enumerated datatype is required, the compiler checks the data and makes sure it is the correct type. For example, if you set an enumerated datatype variable to any other datatype or to an incorrect value, the compiler does not allow it.

CHAPTER 3 Declarations

About this chapter

This chapter explains how to declare variables, constants, and arrays and refer to them in scripts, and how to declare remote procedure calls (RPCs) and external functions that reside in dynamic link libraries (DLLs).

Contents

Declaring variables

General information

Before you use a variable in a PocketBuilder script, you must declare it (give it a datatype and a name).

A variable can be a standard datatype, a structure, or an object. Object datatypes can be system objects as displayed in the Browser or they can be objects you have defined by deriving them from those system object types. For most variables, you can assign it a value when you declare it. You can always assign it a value within a script.

Where to declare variables

Scope

You determine the scope of a PowerScript variable by selecting where you declare it. Instance variables have additional access keywords that restrict specific scripts from accessing the variable.

The following table shows the four scopes of variables.

Table 3-1: PowerScript variable scopes

Scope	Description
Global	Accessible anywhere in the application. It is independent of any object definition.
Instance	Belongs to an object and is associated with an instance of that object (you can think of it as a property of the object). Instance variables have access keywords that determine whether scripts of other objects can access them. They can belong to the application object, a window, a user object, or a menu.
Shared	Belongs to an object definition and exists across all instances of the object. Shared variables retain their value when an object is closed and opened again. Shared variables are always private. They are accessible only in scripts for the object and for controls associated with the object. They can belong to the application object, a window, a user object, or a menu.
Local	A temporary variable that is accessible only in the script in which you define it. When the script has finished executing, the variable constant ceases to exist.

Global, instance, and shared declarations

Global, instance, and shared variables can be defined in the Script view of the Application, Window, User Object, or Menu painters. Global variables can also be defined in the Function painter:

1 Select Declare from the first drop-down list in the Script view.

2 Select the type of variable you want to declare in the second drop-down list of the Script view.

3 Type the declaration in the scripting area of the Script view.

Local declarations

You declare local variables for an object or control in the script for that object or control.

Declaring SQL cursors

You can also declare SQL cursors that are global, shared, instance, or local. Open a specific script or select a variable declaration scope in the Script view and type the DECLARE SQL statement or select Paste SQL from the PainterBar or pop-up menu.

About using variables

General information

To use or set a variable's value in a PocketBuilder script, you name the variable. The variable must be known to the compiler—in other words, it must be in scope.

You can use a variable anywhere you need its value—for example, as a function argument or in an assignment statement.

How PocketBuilder looks for variables

When PocketBuilder executes a script and finds an unqualified reference to a variable, it searches for the variable in the following order:

1. A local variable
2. A shared variable
3. A global variable
4. An instance variable

As soon as PocketBuilder finds a variable with the specified name, it uses the variable's value.

Referring to global variables

To refer to a global variable, you specify its name in a script. However, if the global variable has the same name as a local or shared variable, the local or shared variable will be found first.

To refer to a global variable that is masked by a local or shared variable of the same name, use the global scope operator (::) before the name:

::*globalname*

For example, this statement compares the value of local and global variables, both named total:

```
IF total < ::total THEN ...
```

Referring to instance variables

You can refer to an instance variable in a script if there is an instance of the object open in the application. Depending on the situation, you might need to qualify the name of the instance variable with the name of the object defining it.

Using unqualified names You can refer to instance variables without qualifying them with the object name in the following cases:

- For application-level variables, in scripts for the application object
- For window-level variables, in scripts for the window itself and in scripts for controls in that window

- For user-object-level variables, in scripts for the user object itself and in scripts for controls in that user object
- For menu-level variables, in scripts for a menu object, either the highest-level menu or scripts for the menu objects included as items on the menu

For example, if w_emp has an instance variable *EmpID*, then you can reference *EmpID* without qualification in any script for w_emp or its controls as follows:

```
sle_id.Text = EmpID
```

Using qualified names In all other cases, you need to qualify the name of the instance variable with the name of the object using dot notation:

object.instancevariable

This requirement applies only to Public instance variables. You cannot reference Private instance variables outside the object at all, qualified or not.

For example, to refer to the w_emp instance variable *EmpID* from a script outside the window, you need to qualify the variable with the window name:

```
sle_ID.Text = w_emp.EmpID
```

There is another situation in which references must be qualified. Suppose that w_emp has an instance variable *EmpID* and that in w_emp there is a CommandButton that declares a local variable *EmpID* in its Clicked script. In that script, you must qualify all references to the instance variable:

```
Parent.EmpID
```

Using pronouns as name qualifiers

To avoid ambiguity when referring to variables, you might decide to always use qualified names for object variables. Qualified names leave no doubt about whether a variable is local, instance, or shared.

To write generic code but still use qualified names, you can use the pronouns This and Parent to refer to objects. Pronouns keep a script general by allowing you to refer to the object without naming it specifically.

Window variables in window scripts In a window script, use the pronoun This to qualify the name of a window instance variable. For example, if a window has an instance variable called *index*, then the following statements are equivalent in a script for that window, as long as there is no local or global variable named *index*:

```
index = 5
This.index = 5
```

Window variables in control scripts In a script for a control in a window, use the pronoun Parent to qualify the name of a window instance variable—the window is the parent of the control. In this example, the two statements are equivalent in a script for a control in that window, as long as there is no local or global variable named “index”:

```
index = 5
Parent.index = 5
```

Naming errors If a local or global variable exists with the name “index,” then the unqualified name refers to the local or global variable. It is a programming error if you meant to refer to the object variable. You get an informational message from the compiler if you use the same name for instance and global variables.

Syntax of a variable declaration

Simple syntax

In its simplest form, a PowerScript variable declaration requires only two parts: the datatype and the variable name. For example:

datatype *variablename*

Full syntax

The full syntax allows you to specify access and an initial value. Arrays and some datatypes, such as blobs and decimals, accept additional information:

{ *access* } *datatype* { *{ size }* } { *{ precision }* } *variablename* { = *value* }
{, *variablename2* { = *value2* } }

Table 3-2: Variable declaration parameters

Parameter	Description
access (optional)	(For instance variables only) Keywords specifying the access for the variable. For information, see “Access for instance variables” on page 40.
datatype	The datatype of the variable. You can specify a standard datatype, a system object, or a previously defined structure. For blobs and decimals, you can specify the size or precision of the data by including an optional value in brackets.
{ size } (optional)	(For blobs only) A number, enclosed in braces, specifying the size in bytes of the blob. If *{ size }* is omitted, the blob has an initial size of zero and PocketBuilder adjusts its size each time it is used during execution. If you enter a size that exceeds the declared length in a script, PocketBuilder truncates the blob data.

Parameter	Description
{ precision } (optional)	(For decimals only) A number, enclosed in braces, specifying the number of digits after the decimal point. If you do not specify a precision, the variable takes the precision assigned to it in the script.
variablename	The name of the variable (must be a valid PowerScript identifier, as described in "Identifier names" on page 5). You can define additional variables with the same datatype by naming additional variable names, separated by commas; each variable can have a value.
value (optional)	A literal or expression of the appropriate datatype that will be the initial value of the variable. Blobs cannot be initialized with a value. For information, see "Initial values for variables" on page 38.

Examples

Declaring instance variables

```
integer ii_total = 100 // Total shares
date id_date // Date shares were bought
```

Declaring a global variable

```
string gs_name
```

Declaring shared variables

```
time st_process_start
string ss_process_name
```

Declaring local variables

```
string ls_city = "Boston"
integer li_count
```

Declaring blobs This statement declares *ib_Emp_Picture* a blob with an initial length of zero. The length is adjusted when data is assigned to it:

```
blob ib_Emp_Picture
```

This statement declares *ib_Emp_Picture* a blob with a fixed length of 100 bytes:

```
blob{100} ib_Emp_Picture
```

Declaring decimals These statements declare shared variables *sc_Amount* and *sc_dollars_accumulated* as decimal numbers with two digits after the decimal point:

```
decimal{2} sc_Amount
decimal{2} sc_dollars_accumulated
```

This statement declares *lc_Rate1* and *lc_Rate2* as decimal numbers with four digits after the decimal point:

```
dec{4} lc_Rate1, lc_Rate2
```

This statement declares *lc_Balance* as a decimal with zero digits after the decimal point:

```
decimal{0} lc_Balance
```

This statement does not specify the number of decimal places for *lc_Result*. After the product of *lc_Op1* and *lc_Op2* is assigned to it, *lc_Result* has four decimal places:

```
dec lc_Result
dec{2} lc_Op1, lc_Op2
lc_Result = lc_Op1 * lc_Op2
```

Datatype of a variable

A PowerScript variable can be declared as one of the following datatypes:

- A standard datatype (such as an integer or string).
- An object or control (such as a window or CommandButton).
- An object or structure that you have defined (such as a window called mywindow). An object you have defined must be in a library on the application's library search path when the script is compiled.

Variable names

In a well-planned application, standards determine how you name your PowerScript variables. Naming conventions make scripts easy to understand and help you avoid name conflicts. A typical approach is to include a prefix that identifies the scope and the datatype of the variable. For example, a prefix for an instance variable's name typically begins with *i* (such as *ii_count* or *is_empname*), a local integer variable's name would be *li_total* and a global integer variable's name would be *gi_total*.

For information about naming conventions, see the *Users Guide*.

X and Y as variable names

Although you might think of *x* and *y* as typical variable names, in PocketBuilder they are also properties that specify an object's onscreen coordinates. If you use them as variables and forget to declare them, you do *not* get a compiler error. Instead, PocketBuilder assumes you want to move the object, which might lead to unexpected results in your application.

Initial values for variables

When you declare a PowerScript variable, you can accept the default initial value or specify an initial value in the declaration.

Default values for variables

If you do not initialize a variable when you declare it, PocketBuilder sets the variable to the default value for its datatype as shown in the following table.

Table 3-3: Default initial values for variables

For this variable datatype	PocketBuilder sets this default value
Blob	A blob of 0 length; an empty blob
Char (or character)	ASCII value 0
Boolean	false
Date	1900-01-01 (January 1, 1900)
DateTime	1900-01-01 00:00:00
Numeric (integer, long, longlong, decimal, real, double, UnsignedInteger, and UnsignedLong)	0
String	Empty string ("")
Time	00:00:00 (midnight)

Specifying a literal as a initial value

To initialize a variable when you declare it, place an equal sign (=) and a literal appropriate for that variable datatype after the variable. For information about literals for specific datatypes, see "Standard datatypes" on page 19.

This example declares *li_count* as an integer whose value is 5:

```
integer li_count=5
```

This example declares *li_a* and *li_b* as integers and initializes *li_a* to 5 and *li_b* to 10:

```
integer li_a=5, li_b=10
```

This example initializes *ls_method* with the string "UPS":

```
string ls_method="UPS"
```

This example initializes *ls_headers* to three words separated by tabs:

```
string ls_headers = "Name~tAddress~tCity"
```

This example initializes *li_a* to 1 and *li_c* to 100, leaving *li_b* set to its default value of zero:

```
integer li_a=1, li_b, li_c=100
```

This example declares *ld_StartDate* as a date and initializes it with the date February 1, 1993:

```
date ld_StartDate = 1993-02-01
```

Specifying an expression as an initial value

You can initialize a variable with the value of an existing variable or expression, such as:

```
integer i = 100
integer j = i
```

When you do this, the second variable is initialized with the value of the expression when the script is compiled. The initialization is not reevaluated during execution.

If the expression's value changes Because the expression's value is set to the variable when the script is compiled (not during execution) make sure the expression is not one whose value is based on current conditions. If you want to specify an expression whose value will be different when the application is executed, do not initialize the variable in the declaration. For such values, declare the variable and assign the value in separate statements.

In this declaration, the value of *d_date* is the date *the script is compiled*:

```
date d_date = Today( )
```

In contrast, these statements result in *d_date* being set to the date *the application is run*:

```
date d_date
d_date = Today( )
```

How shared variables are initialized

When you use a shared variable in a script, the variable is initialized when the first instance of the object is opened. When the object is closed, the shared variable continues to exist until you exit the application. If you open the object again without exiting the application, the shared variable will have the value it had when you closed the object.

For example, if you set the shared variable *Count* to 20 in the script for a window, then close the window, and then reopen the window without exiting the application, *Count* will be equal to 20.

When using multiple instances of windows
If you have multiple instances of the window in the example above, *Count* will be equal to 20 in each instance. Since shared variables are shared among all instances of the window, changing *Count* in any instance of the window changes it for all instances.

How instance variables are initialized

When you define an instance variable for a window, menu, or application object, the instance variable is initialized when the object is opened. Its initial value is the default value for its datatype or the value specified in the variable declarations.

When you close the object, the instance variable ceases to exist. If you open the object again, the instance variable is initialized again.

When to use multiple instances of windows When you build a script for one of multiple instances of a window, instance variables can have a different value in each instance of the window. For example, to set a flag based on the contents of the instance of a window, you would use an instance variable.

When to use shared variables instead Use a shared variable instead of an instance variable if you need a variable that:

- Keeps the same value over multiple instances of an object
- Continues to exist after the object is closed

Access for instance variables

Description

The general syntax for declaring PowerScript variables (see "Syntax of a variable declaration" on page 35) showed that you can specify access keywords in a declaration for an instance variable. This section describes those keywords.

When you specify an access right for a variable, you are controlling the visibility of the variable or its visibility access. Access determines which scripts recognize the variable's name.

For a specified access right, you can control operational access with modifier keywords. The modifiers specify which scripts can read the variable's value and which scripts can change it.

Syntax

{ *access-right* } { *readaccess* } { *writeaccess* } *datatype variablename*

The following table describes the parameters you can use to specify access rights for instance variables.

Table 3-4: Instance variable declaration parameters for access rights

Parameter	Description
access-right (optional)	A keyword specifying where the variable’s name will be recognized. Values are: • PUBLIC — (Default) Any script in the application can refer to the variable. In another object’s script, you use dot notation to qualify the variable name and identify the object it belongs to. • PROTECTED — Scripts for the object for which the variable is declared and its descendants can refer to the variable. • PRIVATE — Scripts for the object for which the variable is declared can refer to the variable. You cannot refer to the variable in descendants of the object.
readaccess (optional)	A keyword restricting the ability of scripts to read the variable’s value. Values are: • PROTECTEDREAD — Only scripts for the object and its descendants can read the variable. • PRIVATEREAD — Only scripts for the object can read the variable. When *access-right* is PUBLIC, you can specify either keyword. When *access-right* is PROTECTED, you can specify only PRIVATEREAD. You cannot specify a modifier for PRIVATE access, because PRIVATE is already fully restricted. If *readaccess* is omitted, any script can read the variable.
writeaccess (optional)	A keyword restricting the ability of scripts to change the variable’s value. Values are: • PROTECTEDWRITE — Only scripts for the object and its descendants can change the variable. • PRIVATEWRITE — Only scripts for the object can change the variable. When *access-right* is PUBLIC, you can specify either keyword. When *access-right* is PROTECTED, you can specify only PRIVATEWRITE. You cannot specify a modifier for PRIVATE access, because PRIVATE is already fully restricted. If *writeaccess* is omitted, any script can change the variable.

Parameter	Description
datatype	A valid datatype. See “Syntax of a variable declaration” on page 35.
variablename	A valid identifier. See “Syntax of a variable declaration” on page 35.

Usage

Access modifiers give you more control over which objects have access to a particular object’s variables. A typical use is to declare a public variable but only allow the owner object to modify it:

```
public protectedwrite integer ii_count
```

You can also group declarations that have the same access by specifying the access-right keyword as a label (see "Another format for access-right keywords" next).

When you look at exported object syntax, you might see the access modifiers SYSTEMREAD and SYSTEMWRITE. Only PocketBuilder can access variables with these modifiers. You cannot refer to variables with these modifiers in your scripts and functions and you cannot use these modifiers in your own definitions.

Examples

To declare these variables, select Declare>Instance Variables in the appropriate painter.

These declarations use access keywords to control the scripts that have access to the variables:

```
private integer ii_a, ii_n
public  integer ii_Subtotal
protected integer ii_WinCount
```

This protected variable can only be changed by scripts of the owner object; descendants of the owner can read it:

```
protected privatewrite string is_label
```

These declarations have public access (the default) but can only be changed by scripts in the object itself:

```
privatewrite real ir_accum, ir_current_data
```

This declaration defines an integer that only the owner objects can write or read but whose name is reserved at the public level:

```
public privateread privatewrite integer ii_reserved
```

Private variable not recognized outside its object Suppose you have defined a window w_emp with a private integer variable ii_int:

```
private integer ii_int
```

In a script you declare an instance of the window called w_myemp. If you refer to the private variable *ii_int*, you get a compiler warning that the variable is not defined (because the variable is private and is not recognized in scripts outside the window itself):

```
w_emp w_myemp
w_myemp.ii_int = 1 // Variable not defined
```

Public variable with restricted access Suppose you have defined a window w_emp with a public integer variable *ii_int* with write access restricted to private:

```
public privatewrite integer ii_int
```

If you write the same script as above, the compiler warning will say that you cannot write to the variable (the name is recognized because it is public, but write access is not allowed):

```
w_emp w_myemp
w_myemp.ii_int = 1 // Cannot write to variable
```

Another format for access-right keywords

Description

You can also group declarations of PowerScript variables according to access by specifying the access-right keyword as a label. It appears on its own line, followed by a colon (:).

Syntax

access-right:

{ *readaccess* } { *writeaccess* } *datatype variablename*

{ *access-right* } { *readaccess* } { *writeaccess* } *datatype variablename*

{ *readaccess* } { *writeaccess* } *datatype variablename*

Within a labeled group of declarations, you can override the access on a single line by specifying another access-right keyword with the declaration. The labeled access takes effect again on the following lines.

Examples

In these declarations, the instance variables have the access specified by the label that precedes them. Another private variable is defined at the end, where private overrides the public label:

```
Private:
integer ii_a=10, ii_b=24
string  is_Name, is_Address1
```

```
Protected:
integer ii_Units
double  idb_Results
string  is_Lname
Public:
integer ii_Weight
string  is_Location="Home"
private integer ii_test
```

Some of these protected declarations have restricted write access:

```
Protected:
integer ii_Units
privatewrite double  idb_Results
privatewrite string  is_Lname
```

Declaring constants

Description

Any PowerScript variable declaration of a standard datatype that can be assigned an initial value can be a constant instead of a variable. To make it a constant, include the keyword CONSTANT in the declaration and assign it an initial value.

Syntax

CONSTANT { *access* } *datatype constname* = *value*

The following table shows the parameters used to declare constants.

Table 3-5: Constant variable declaration parameters

Parameter	Description
CONSTANT	Declares a constant instead of a variable. The CONSTANT keyword can be before or after the *access* keywords.
access (optional)	(For instance variables only) Keywords specifying the access for the constant. For information, see “Access for instance variables” on page 40.
datatype	A standard datatype for the constant. For decimals, you can include an optional value in brackets to specify the precision of the data. Blobs cannot be constants. For information about PocketBuilder datatypes, see “Standard datatypes” on page 19.

Parameter	Description
constname	The name of the constant (must be a valid PowerScript identifier, as described in "Identifier names" on page 5).
value	A literal or expression of the appropriate datatype that will be the value of the constant. The value is required. For information, see "Initial values for variables" on page 38.

Usage

When declaring a constant, an initial value is required. Otherwise, a compiler error occurs. Assigning a value to a constant after it is declared (that is, redefining a constant in a descendant object) also causes a compiler error.

Examples

Although PowerScript is not case sensitive, these examples of local constants use a convention of capitalizing constant names:

```
constant string LS_HOMECITY = "Boston"
constant real LR_PI = 3.14159265
```

Declaring arrays

Description

An array is an indexed collection of elements of a single datatype. In PocketBuilder, an array can have one or more dimensions. One-dimensional arrays can have a fixed or variable size; multidimensional arrays always have a fixed size. Each dimension of an array can have 2,147,483,647 bytes of elements.

Any simple variable declaration becomes an array when you specify brackets after the variable name. For fixed-size arrays, you specify the sizes of the dimensions inside those brackets.

Syntax

{ *access* } *datatype variablename* { *d1*, ..., *dn* } { = { *valuelist* } }

The following table describes the parameters used to declare array variables.

Table 3-6: Array variable declaration parameters

Parameter	Description
access (optional)	(For instance variables only) Keywords specifying the access for the variable. For information, see "Access for instance variables" on page 40.

Parameter	Description
datatype	The datatype of the variable. You can specify a standard datatype, a system object, or a previously defined structure. For decimals, you can specify the precision of the data by including an optional value in brackets after *datatype* (see "Syntax of a variable declaration" on page 35): `decimal {2} variablename [ ]` For blobs, fixed-length blobs within an array are not supported. If you specify a size after *datatype*, it is ignored.
variablename	The name of the variable (name must be a valid PowerScript identifier, as described in "Identifier names" on page 5). You can define additional arrays with the same datatype by naming additional variable names with brackets and optional value lists, separated by commas.
[{ *d1*, ..., *dn* }]	Brackets and (for fixed-size arrays) one or more integer values (*d1* through *dn*, one for each dimension) specifying the sizes of the dimensions. For a variable-size array, which is always one-dimensional, specify brackets only. For more information on how variable-size arrays change size, see "Size of variable-size arrays" on page 50. For a fixed-size array, the number of dimensions is determined by the number of integers you specify and is limited only by the amount of available memory. For fixed-size arrays, you can use TO to specify a range of element numbers (instead of a dimension size) for one or more of the dimensions. Specifying TO allows you to change the lower bound of the dimension (*upperbound* must be greater than *lowbound*): [*d1lowbound* TO *d1upperbound* {, ... , *dnlowbound* TO *dnupperbound* }]
{ *valuelist* } (optional)	A list of initial values for each position of the array. The values are separated by commas and the whole list is enclosed in braces. The number of values cannot be greater than the number of positions in the array. The datatype of the values must match *datatype*.

Examples

These declarations create variable-size arrays:

```
integer li_stats[ ]        // Array of integers.
decimal {2} ld_prices[ ]   // Array of decimals with
                              // 2 places of precision.
blob lb_data[ ]            // Array of variable-size
                              // blobs.
date ld_birthdays[ ]       // Array of dates.
string ls_city[ ]         // Array of strings.
                              // Each string can be
                              // any length.
```

This statement declares a variable-size array of decimal number (the declaration does not specify a precision, so each element in the array takes the precision of the value assigned to it):

```
dec lc_limit[ ]
```

Fixed arrays These declarations create fixed-size, one-dimensional arrays:

```
integer li_TaxCode[3]  // Array of 3 integers.
string ls_day[7]       // Array of 7 strings.
blob ib_image[10]      // Array of 10
                       // variable-size blobs.
dec{2} lc_Cost[10]     // Array of 10 decimal
                       // numbers.
                       // Each value has 2 digits
                       // following the decimal
                       // point.
decimal lc_price[20]   // Array of 20 decimal
                       // numbers.
                       // Each takes the precision
                       // of the value assigned.
```

Using TO to change array index values These fixed-size arrays use TO to change the range of index values for the array:

```
real lr_Rate[2 to 5]      // Array of 4 real numbers:
                           // Rate[2] through Rate[5]
integer li_Qty[0 to 2]     // Array of 3 integers
string ls_Test[-2 to 2]    // Array of 5 strings
integer li_year[76 to 96]  // Array of 21 integers
string ls_name[-10 to 15]  // Array of 26 strings
```

Incorrect declarations using TO In an array dimension, the second number must be greater than the first. These declarations are invalid:

```
integer li_count[10 to 5]     // INVALID: 10 is
                              // greater than 5
```

```
integer li_price[-10 to -20]  // INVALID: -10
                              // is greater than -20
```

Arrays with two or more dimensions This declaration creates a six-element, two-dimensional integer array. The individual elements are *li_score[1,1]*, *li_score[1,2]*, *li_score[1,3]*, *li_score[2,1]*, *li_score[2,2]*, and *li_score[2,3]*:

```
integer li_score[2,3]
```

This declaration specifies that the indexes for the dimensions are 1 to 5 and 10 to 25:

```
integer li_RunRate[1 to 5, 10 to 25]
```

This declaration creates a 3-dimensional 45,000-element array:

```
long ll_days[3, 300, 50]
```

This declaration changes the subscript range for the second and third dimension:

```
integer li_staff[100, 0 to 20, -5 to 5]
```

More declarations of multidimensional arrays:

```
string ls_plant[3,10]   // two-dimensional array
                        // of 30 strings
dec{2} lc_rate[3,4]     // two-dimensional array of 12
                        // decimals with 2 digits
                        // after the decimal point
```

This declaration creates three decimal arrays:

```
decimal{3} lc_first[10],lc_second[15,5],lc_third[ ]
```

Values for array elements

General information

PocketBuilder initializes each element of an array to the same default value as its underlying datatype. For example, in a newly declared integer array:

```
integer li_TaxCode[3]
```

the elements *li_TaxCode[1]*, *li_TaxCode[2]*, and *li_TaxCode[3]* are all initialized to zero.

For information about default values for basic datatypes, see “Initial values for variables” on page 38.

Simple array

In a simple array, you can override the default values by initializing the elements of the array when you declare the array. You specify the values in a comma-separated list of values enclosed in braces. You do not have to initialize all the elements of the array, but you cannot initialize values in the middle or end without initializing the first elements.

Multidimensional array

In a multidimensional array, you still provide the values in a simple, comma-separated list. When the values are assigned to array positions, the first dimension is the fastest-varying dimension, and the last dimension is the slowest-varying. In other words, the values are assigned to array positions by looping over all the values of the first dimension for each value of the second dimension, then looping over all the values of the second dimension for each value of the third, and so on.

Assigning values

You can assign values to an array after declaring it using the same syntax of a list of values within braces:

```
integer li_Arr[]
Li_Arr = {1, 2, 3, 4}
```

Examples

Example 1 This statement declares an initialized one-dimensional array of three variables:

```
real lr_Rate[3]={1.20, 2.40, 4.80}
```

Example 2 This statement initializes a two-dimensional array:

```
integer li_units[3,4] = {1,2,3, 1,2,3, 1,2,3, 1,2,3}
```

As a result:

Li_units[1,1], *[1,2]*, *[1,3]*, and *[1,4]* are all 1
Li_units[2,1], *[2,2]*, *[2,3]*, and *[2,4]* are all 2
Li_units[3,1], *[3,2]*, *[3,3]*, and *[3,4]* are all 3

Example 3 This statement initializes the first half of a 3-dimensional array:

```
integer li_units[3,4,2] = &
 {1,2,3, 1,2,3, 1,2,3, 1,2,3}
```

As a result:

Li_units[1,1,1], *[1,2,1]*, *[1,3,1]*, and *[1,4,1]* are all 1
Li_units[2,1,1], *[2,2,1]*, *[2,3,1]*, and *[2,4,1]* are all 2
Li_units[3,1,1], *[3,2,1]*, *[3,3,1]*, and *[3,4,1]* are all 3
Li_units[1,1,2], *[1,2,2]*, *[1,3,2]*, and *[1,4,2]* are all 0
Li_units[2,1,2], *[2,2,2]*, *[2,3,2]*, and *[2,4,2]* are all 0
Li_units[3,1,2], *[3,2,2]*, *[3,3,2]*, and *[3,4,2]* are all 0

Size of variable-size arrays

General information

A variable-size array consists of a variable name followed by square brackets but no number. PocketBuilder defines the array elements *by use* at execution time (subject only to memory constraints). Only one-dimensional arrays can be variable-size arrays.

Because you do not declare the size, you cannot use the TO notation to change the lower bound of the array, so the lower bound of a variable-size array is always 1.

How memory is allocated

Initializing elements of a variable-size array allocates memory for those elements. You specify initial values just as you do for fixed-size arrays, by listing the values in braces. The following statement sets *code[1]* equal to 11, *code[2]* equal to 242, and *code[3]* equal to 27. The array has a size of 3 initially, but the size will change if you assign values to higher positions:

```
integer li_code[ ]={11,242,27}
```

For example, these statements declare a variable-size array and assigns values to three array elements:

```
long ll_price[ ]
ll_price[100] = 2000
ll_price[50]  = 3000
ll_price[110] = 5000
```

When these statements first execute, they allocate memory as follows:

- The statement `ll_price[100]=2000` will allocate memory for 100 long numbers *ll_price[1]* to *ll_price[100]*, then assign 0 (the default for numbers) to *ll_price[1]* through *ll_price[99]* and assign 2000 to *ll_price[100]*.
- The statement `ll_price[50]=3000` will not allocate more memory but will assign the value 3000 to the 50th element of the *ll_price* array.
- The statement `ll_price[110]=5000` will allocate memory for 10 more long numbers named *ll_price[101]* to *ll_price[110]* and then assign 0 (the default for numbers) to *ll_price[101]* through *ll_price[109]* and assign 5000 to *ll_price[110]*.

More about arrays

This section provides technical details about:

- Assigning one array to another

- Using arraylists to assign values to an array
- Errors that occur when addressing arrays

Assigning one array to another

General information

When you assign one array to another, PocketBuilder uses the following rules to map the values of one onto the other.

One-dimensional arrays

To an unbounded array The target array is the same as the source:

```
integer a[ ], b[ ]
a = {1,2,3,4}
b = a
```

To a bounded array If the source array is smaller, values from the source array are copied to the target array and extra values are set to zero. In this example, *b[5]* and *b[6]* are set to 0:

```
integer a[ ], b[6]
a = {1,2,3,4}
b = a
```

If the source array is larger, values from the source array are copied to the target array until it is full (and extra values from the source array are ignored). In this example, the array *b* has only the first three elements of *a*:

```
integer a[ ], b[3]
a = {1,2,3,4}
b = a
```

Multidimensional arrays

PocketBuilder stores multidimensional arrays in column major order, meaning the first subscript is the fastest varying—[1,1], [2,1], [3,1]).

When you assign one array to another, PocketBuilder linearizes the source array in column major order, making it a one-dimensional array. PocketBuilder then uses the rules for one-dimensional arrays (described above) to assign the array to the target.

Not all array assignments are allowed, as described in the following rules.

One multidimensional array to another If the dimensions of the two arrays match, the target array becomes an exact copy of the source:

```
integer a[2,10], b[2,10]
a = b
```

If both source and target are multidimensional but do not have matching dimensions, the assignment is not allowed and the compiler reports an error:

```
integer a[2,10], b[4,10]
a = b // Compiler error
```

One-dimensional array to a multidimensional array A one-dimensional array can be assigned to a multidimensional array. The values are mapped onto the multidimensional array in column major order:

```
integer a[ ], b[2,2]
b = a
```

Multidimensional array to a one-dimensional array A multidimensional array can also be assigned to a one-dimensional array. The source is linearized in column major order and assigned to the target:

```
integer a[ ], b[2,2]
a = b
```

Examples

Suppose you declare three arrays (*a*, *b*, and *c*). One (*c*) is unbounded and one-dimensional; the other two (*a* and *b*) are multidimensional with different dimensions:

```
integer c[ ], a[2,2], b[3,3] = {1,2,3,4,5,6,7,8,9}
```

Array *b* is laid out like this:

`1 for b[1,1]`	`4 for b[1,2]`	`7 for b[1,3]`
`2 for b[2,1]`	`5 for b[2,2]`	`8 for b[2,3]`
`3 for b[3,1]`	`6 for b[3,2]`	`9 for b[3,3]`

This statement causes a compiler error, because *a* and *b* have different dimensions:

```
a = b // Compiler error
```

This statement explicitly linearizes *b* into *c*:

```
c = b
```

You can then assign the linearized version of the array to *a*:

```
a = c
```

The values in array *a* are laid out like this:

`1 for a[1,1]`	`3 for a[1,2]`
`2 for a[2,1]`	`4 for a[2,2]`

Initializing *a* with an arraylist produces the same result:

```
integer a[2,2] = {1,2,3,4}
```

The following section describes arraylists.

Using arraylists to assign values to an array

General information

In PocketBuilder, an arraylist is a list of values enclosed in braces used to initialize arrays. An arraylist represents a one-dimensional array, and its values are assigned to the target array using the rules for assigning arrays described in "Assigning one array to another" on page 51.

Examples

In this declaration, a variable-size array is initialized with four values:

```
integer a[ ] = {1,2,3,4}
```

In this declaration, a fixed-size array is initialized with four values (the rest of its values are zeros):

```
integer a[10] = {1,2,3,4}
```

In this declaration, a fixed-size array is initialized with four values. Because the array's size is set at 4, the rest of the values in the arraylist are ignored:

```
integer a[4] = {1,2,3,4,5,6,7,8}
```

In this declaration, values 1, 2, and 3 are assigned to the first column and the rest to the second column:

```
integer a[3,2] = {1,2,3,4,5,6}
```

1	4
2	5
3	6

If you think of a three-dimensional array as having pages of rows and columns, then the first column of the first page has the values 1 and 2, the second column on the first page has 3 and 4, and the first column on the second page has 5 and 6.

The second column on the second page has zeros:

```
integer a[2,2,2] = {1,2,3,4,5,6}
```

1	3		5	0
2	4		6	0

Errors that occur when addressing arrays

Fixed-size arrays

In PocketBuilder, referring to array elements outside the declared size causes an error during execution; for example:

```
int test[10]
test[11]=50      // This causes an execution error.
test[0]=50       // This causes an execution error.
int trial[5,10]
trial [6,2]=75   // This causes an execution error.
trial [4,11]=75  // This causes an execution error.
```

Variable-size arrays

Assigning a value to an element of a variable-size array that is outside its current values increases the array's size. However, accessing a variable-size array above its largest assigned value or below its lower bound causes an error during execution:

```
integer li_stock[ ]
li_stock[50]=200
          // Establish array size 50 elements.
IF li_stock[51]=0 then Beep(1)
          // This causes an execution error.
IF li_stock[0]=0 then Beep(1)
          // This causes an execution error.
```

Declaring external functions

Description

External functions are functions written in languages other than PowerScript and stored in dynamic link libraries. On Windows and Windows CE, dynamic libraries have the extension *DLL*. You can use external functions that are written in any language that supports dynamic libraries.

Before you can use an external function in a script, you must declare it as one of two types:

- **Global external functions** These are available anywhere in the application.
- **Local external functions** These are defined for a particular type of window, menu, user object, or user-defined function. These functions are part of the object's definition and can always be used in scripts for the object itself. You can also choose to make these functions accessible to other scripts.

To understand how to declare and call an external function, see the documentation from the developer of the external function library.

Syntax

External function syntax Use the following syntax to declare an external function:

{ *access* } FUNCTION *returndatatype name* ({ { REF } *datatype1 arg1*,
 ..., { REF } *datatypen argn* }) LIBRARY "*libname*"
 ALIAS FOR "*extname*"

External subroutine syntax To declare external subroutines (which are the same as external functions except that they do not return a value), use this syntax:

{ *access* } SUBROUTINE *name* ({ { REF } *datatype1 arg1*, ...,
 { REF } datatypen argn }) LIBRARY "*libname*"
 ALIAS FOR "*extname*"

The following table describes the parameters used to declare external functions and subroutines:

Table 3-7: External function or subroutine declaration parameters

Parameter	Description
access (optional)	(Local external functions only) Public, Protected, or Private specifies the access level of a local external function. The default is Public. For more information, see the section about specifying access of local functions in "Usage" next.
FUNCTION or SUBROUTINE	A keyword specifying the type of call, which determines the way return values are handled. If there is a return value, declare it as a FUNCTION; if it returns nothing or returns VOID, specify SUBROUTINE.
returndatatype	The datatype of the value returned by the function.
name	The name of a function or subroutine that resides in a DLL.
REF	A keyword that specifies that you are passing by reference the argument that follows REF. The function can store a value in *arg* that will be accessible to the rest of the PocketBuilder script.
datatype arg	The datatype and name of the arguments for the function or subroutine. The list must match the definition of the function in the DLL. Each *datatype arg* pair can be preceded by REF. For more information on passing arguments, see the *Resource Guide* or see *Application Techniques* in the PowerBuilder documentation set.

Parameter	Description
LIBRARY "*libname*"	A keyword followed by a string containing the name of the dynamic library in which the function or subroutine is stored. *libname* is a dynamic link library, which is a file that usually has the extension *DLL*.
ALIAS FOR "*extname*" (optional)	Keywords followed by a string giving the name of the function as defined in the dynamic library. If the name in the dynamic library is not the name you want to use in your script, or if the name in the database is not a legal PowerScript name, you must specify ALIAS FOR "*extname*" to establish the association between the PowerScript name and the external name.

Usage

Specifying access of local functions When declaring a local external function, you can specify its access level—which scripts have access to the function.

The following table describes where local external functions can be used when they are declared with a given access level:

Table 3-8: Access levels for local external functions

Access level	Where you can use the local external function
Public	Any script in the application.
Private	Scripts for events in the object for which the function is declared. You cannot use the function in descendants of the object.
Protected	Scripts for the object for which the function is declared and its descendants.

Use of the access keyword with local external functions works the same as the access-right keywords for instance variables.

Availability of the dynamic library during execution

To be available to a PocketBuilder application running on any Windows CE platform, the DLL must be in one of the following directories:

- The current directory
- The Windows directory

For PowerBuilder applications running on the desktop, the DLL can also be in one of the following directories:

- The Windows System subdirectory
- Directories on the DOS path

Examples

In the examples application that comes with PowerBuilder, external functions are declared as local external functions in a user object called u_external_function_win32. The scripts that call the functions are user object functions, but because they are part of the same user object, you do not need to use object notation to call them.

Example 1 These declarations allow PowerBuilder to call the functions required for playing a sound in the *WINMM.DLL*:

```
//playsound
FUNCTION boolean sndPlaySoundA (string SoundName,
   uint Flags) LIBRARY "WINMM.DLL"
FUNCTION uint waveOutGetNumDevs () LIBRARY "WINMM.DLL"
```

A function called uf_playsound in the examples application provided with PowerBuilder calls the external functions. Uf_playsound is called with two arguments (*as_filename* and *ai_option*) that are passed through to sndPlaySoundA. Values for *ai_option* are as defined in the Windows documentation, as commented here:

```
//Options as defined in mmystem.h.
//These may be or'd together.
//#define SND_SYNC 0x0000
//play synchronously (default)
//#define SND_ASYNC 0x0001
//play asynchronously
//#define SND_NODEFAULT 0x0002
//do not use default sound
//#define SND_MEMORY 0x0004
//lpszSoundName points to a memory file
//#define SND_LOOP 0x0008
//loop the sound until next sndPlaySound
//#define SND_NOSTOP 0x0010
//do not stop any currently playing sound

uint lui_numdevs

lui_numdevs = WaveOutGetNumDevs()
IF lui_numdevs > 0 THEN
      sndPlaySoundA(as_filename,ai_option)
      RETURN 1
ELSE
      RETURN -1
END IF
```

Example 2 This is the declaration for the Windows GetSysColor function:

```
FUNCTION ulong GetSysColor (int index) LIBRARY
"USER32.DLL"
```

This statement calls the external function. The meanings of the index argument and the return value are specified in the Windows documentation:

```
RETURN GetSysColor (ai_index)
```

Example 3 This is the declaration for the Windows GetSysColor function:

```
FUNCTION int GetSystemMetrics (int index) LIBRARY
"USER32.DLL"
```

These statements call the external function to get the screen height and width:

```
RETURN GetSystemMetrics(1)
RETURN GetSystemMetrics(0)
```

Datatypes for external function arguments

When you declare an external function in PocketBuilder, the datatypes of the arguments must correspond with the datatypes as declared in the function's source definition. This section documents the correspondence between datatypes in external functions and datatypes in PocketBuilder. It also includes information on byte alignment when passing structures by value.

Use the tables to find out what PocketBuilder datatype to use in an external function declaration. The PocketBuilder datatype you select depends on the datatype in the source code for the function. The first column lists datatypes in source code. The second column describes the datatype so you know exactly what it is. The third column lists the PocketBuilder datatype you should use in the external function declaration.

Boolean

BOOL on Windows is 16-bit, signed. It is declared in PocketBuilder as boolean.

Pointers

Table 3-9: PocketBuilder datatypes for pointers

Datatype in source code	Size	PocketBuilder datatype
* (any pointer)	32-bit pointer	Long
byte *	Array of bytes of variable length	Blob

Windows 32-bit FAR pointers, such as LPBYTE, LPDWORD, LPINT, LPLONG, LPVOID, and LPWORD, are declared in PocketBuilder as long datatypes. HANDLE is defined as 32 bits unsigned and is declared in PocketBuilder as an UnsignedLong.

Near-pointer datatypes (such as PSTR and NPSTR) are not supported in PocketBuilder.

Characters and strings

Table 3-10: PocketBuilder datatypes for characters and strings

Datatype in source code	Size	PocketBuilder datatype
char	16 bit Unicode	Char
string	32-bit pointer to a null-terminated array of Unicode characters of variable length	String

The Windows 32-bit FAR pointer LPSTR is declared in PocketBuilder as string.

Reference arguments
When you pass a string to an external function by reference, all memory management is done in PocketBuilder. The string variable must be long enough to hold the returned value. To ensure that this is true, first declare the string variable, and then use the Space function to fill the variable with blanks equal to the maximum number of characters that you expect the function to return.

Fixed-point values

Table 3-11: PocketBuilder datatypes for fixed-point values

Datatype in source code	Size	PocketBuilder datatype
short	16 bits, signed	Integer
unsigned short	16 bits, unsigned	UnsignedInteger
int	32 bits, signed	Long
unsigned int	32 bits, unsigned	UnsignedLong
long	32 bits, signed	Long
unsigned long	32 bits, unsigned	UnsignedLong
longlong	64 bits, signed	LongLong

The Windows definition WORD is declared in PocketBuilder as UnsignedInteger and the Windows definition DWORD is declared as an UnsignedLong. You cannot call external functions with return values or arguments of type short.

Floating-point values

Table 3-12: PocketBuilder datatypes for floating-point values

Datatype in source code	Size and precision	PocketBuilder datatype
float	32 bits, single precision	Real
double	64 bits, double precision	Double

PocketBuilder does not support 80-bit doubles on Windows.

Date and time

The PocketBuilder datatypes Date, DateTime, and Time are structures and have no direct equivalent for external functions in C.

Passing structures by value

You can pass PocketBuilder structures to external C functions if they have the same definitions and alignment as the structure's components. The DLL or shared library must be compiled using byte alignment; no padding is added to align fields within the structure.

Calling external functions

Global external functions

In PocketBuilder, you call global external functions using the same syntax as for calling user-defined global and system functions. As with other global functions, global external functions can be triggered or posted but not called dynamically.

Local external functions

Call local functions using the same syntax as for calling object functions. They can be triggered or posted and called dynamically.

For information

For information, see "Syntax for calling PocketBuilder functions and events" on page 104.

Defining source for external functions

You can use external functions written in any language that supports the standard calling sequence for 32-bit platforms. If you are calling functions on Windows in libraries that you have written yourself, remember that you need to export the functions. Depending on your compiler, you can do this in the function prototype or in a linker definition (*.DEF*) file. For more information about using external functions, see the *Resource Guide* or see *Application Techniques* in the PowerBuilder documentation set.

Declaring DBMS stored procedures as remote procedure calls

Description

In PowerBuilder, you can use dot notation for calling non-result-set stored procedures as remote procedure calls (RPCs):

object.function

You can call database procedures in Sybase, Oracle, Informix, and other ODBC databases with stored procedures.

RPCs provide support for Oracle PL/SQL tables and parameters that are defined as both input and output. You can call overloaded procedures.

Applies to

Transaction object

Syntax

FUNCTION *rtndatatype functionname* ({ { REF } *datatype1 arg1*,...,
    { REF } *datatypen argn* }) RPCFUNC { ALIAS FOR "*spname*" }

SUBROUTINE *functionname* ({ { REF } *datatype1 arg1* , ...,
    { REF } *datatypen argn* }) RPCFUNC { ALIAS FOR "*spname*" }

Table 3-13: RPC declaration parameters

Argument	Description
FUNCTION or SUBROUTINE	A keyword specifying the type of call, which determines the way return values are handled. If there is a return value, declare it as a FUNCTION. If it returns nothing or returns VOID, specify SUBROUTINE.
rtndatatype	In a FUNCTION declaration, the datatype of the value returned by the function.
functionname	The name of the database procedure as you will call it in PowerBuilder. If the name in the DBMS is different, use ALIAS FOR to associate the DBMS name with the PowerBuilder name.
REF	Specifies that you are passing by reference the argument that follows REF. The stored procedure can store a value in *arg* that will be accessible to the rest of the PowerBuilder script. When you pass a string by reference, all memory management is done in PowerBuilder. The string variable must be long enough to hold the returned value. To ensure that this is true, first declare the string variable, and then use the Space function to fill the variable with blanks equal to the maximum number of characters that you expect the function to return.
datatype arg	The datatype and name of the arguments for the stored procedure. The list must match the definition of the stored procedure in the database. Each *datatype arg* pair can be preceded by REF.

Argument	Description
RPCFUNC	A keyword indicating that this declaration is for a stored procedure in a DBMS, not an external function in a DLL. For information on declaring external functions, see "Declaring external functions" on page 54.
ALIAS FOR *"spname"* (optional)	Keywords followed by a string naming the procedure in the database. If the name in the database is not the name you want to use in your script or if the name in the database is not a legal PowerScript name, you must specify ALIAS FOR *"spname"* to establish the association between the PowerScript name and the database name.

Usage

If a function does not return a value (for example, it returns Void), specify the declaration as a subroutine instead of a function.

RPC declarations are always associated with a transaction object. You declare them as local external functions. The Declare Local External Functions dialog box has a Procedures button (if the connected database supports stored procedures), which gives you access to a list of stored procedures in the database.

For more information, see the *Resource Guide*.

Examples

Example 1 This declaration of the GIVE_RAISE_PROC stored procedure is declared in the User Object painter for a transaction object (the declaration appears on one line):

```
FUNCTION double GIVE_RAISE(ref double SALARY) RPCFUNC
ALIAS FOR "GIVE_RAISE_PROC"
```

This code calls the function in a script:

```
double val = 20000
double rv
rv = SQLCA.give_raise(val)
```

Example 2 This declaration for the stored procedure SPM8 does not need an ALIAS FOR phrase, because the PowerBuilder and DBMS names are the same:

```
FUNCTION integer SPM8(integer value) RPCFUNC
```

This code calls the SPM8 stored procedure:

```
int myresult
myresult = SQLCA.spm8(myresult)
IF SQLCA.sqlcode <> 0 THEN
      messagebox("Error", SQLCA.sqlerrtext)
END IF
```

CHAPTER 4

Operators and Expressions

About this chapter

This chapter describes the operators supported in PowerScript and how to use them in expressions.

Contents

Operators in PocketBuilder

General information

Operators perform arithmetic calculations; compare numbers, text, and boolean values; execute relational operations on boolean values; and concatenate strings and blobs.

Three types

PowerScript supports three types of operators:

- Arithmetic operators for numeric datatypes
- Relational operators for all datatypes
- Concatenation operator for string datatypes

Operators used in DataWindow objects
The documentation for DataWindows describes how operators are used in DataWindow expressions.

Arithmetic operators in PocketBuilder

Description

The following table lists the arithmetic operators used in PocketBuilder.

Table 4-1: PocketBuilder arithmetic operators

Operator	Meaning	Example
`+`	Addition	`Total=SubTotal+Tax`
`-`	Subtraction	`Price=Price–Discount` Unless you have prohibited the use of dashes in identifier names, you must surround the minus sign with spaces.
`*`	Multiplication	`Total=Quantity*Price`
`/`	Division	`Factor=Discount/Price`
`^`	Exponentiation	`Rank=Rating^2.5`

Usage

Operator shortcuts for assignments For information about shortcuts that combine arithmetic operators with assignments (such as ++ and +=), see Assignment on page 113.

Subtraction If the option Allow Dashes in Identifiers is checked on the Script tab in the Options dialog box, you must always surround the subtraction operator and the -- operator with spaces. Otherwise, PocketBuilder interprets the expression as an identifier.

For information about dashes in identifiers, see “Identifier names” on page 5.

Multiplication and division Multiplication and division are carried out to full precision (16–18 digits). Decimal numbers are rounded (not truncated) on assignment.

Calculation with NULL When you form an arithmetic expression that contains a NULL value, the expression’s value is null. Thinking of null as *undefined* makes this easier to understand.

For more information about null values, see “NULL values” on page 8.

Errors and overflows The following problems can occur when using arithmetic operators:

- Division by zero, exponentiation of negative values, and so on cause errors during execution
- Overflow of real, double, and decimal values causes errors during execution.

- Overflow of signed or unsigned integers and longs causes results to wrap. However, because integers are promoted to longs in calculations, wrapping does not occur until the result is explicitly assigned to an integer variable.

  For more information about type promotion, see "Datatype of PocketBuilder expressions" on page 69.

Examples

Subtraction This statement always means subtract B from A:

```
A - B
```

If DashesInIdentifiers is set to 1, the following statement means a variable named A-B, but if DashesInIdentifiers is set to 0, it means subtract B from A:

```
A-B
```

Precision for division These examples show the values that result from various operations on decimal values:

```
decimal {4} a,b,d,e,f
decimal {3} c
a = 20.0/3                 // a contains  6.6667
b = 3 * a                  // b contains 20.0001
c = 3 * a                  // c contains 20.000
d = 3 * (20.0/3)           // d contains 20.0000
e = Truncate(20.0/3, 4)     // e contains  6.6666
f = Truncate(20.0/3, 5)     // f contains  6.6667
```

Calculations with null When the value of variable *c* is null, the following assignment statements all set the variable a to null:

```
integer a, b=100, c

SetNULL(c)

a = b+c   // all statements set a to NULL
a = b - c
a = b*c
a = b/c
```

Overflow This example illustrates the value of the variable *i* after overflow occurs:

```
integer i
i = 32767
i = i + 1     // i is now -32768
```

Relational operators in PocketBuilder

Description

PocketBuilder uses relational operators in boolean expressions to evaluate two or more operands. Logical operators can join relational expressions to form more complex boolean expressions.

The result of evaluating a boolean expression is always true or false.

The following table lists relational and logical operators.

Table 4-2: PocketBuilder relational and logical operators

Operator	Meaning	Example
=	Equals	`if Price=100 then Rate=.05`
>	Greater than	`if Price>100 then Rate=.05`
<	Less than	`if Price<100 then Rate=.05`
<>	Not equal	`if Price<>100 then Rate=.05`
>=	Greater than or equal	`if Price>=100 then Rate=.05`
<=	Less than or equal	`if Price<=100 then Rate=.05`
NOT	Logical negation	`if NOT Price=100 then Rate=.05`
AND	Logical and	`if Tax>3 AND Ship <5 then Rate=.05`
OR	Logical or	`if Tax>3 OR Ship<5 then Rate=.05`

Usage

Comparing strings When PocketBuilder compares strings, the comparison is case sensitive. Trailing blanks are significant.

For information on comparing strings regardless of case, see the functions Upper on page 1056 and Lower on page 684.

To remove trailing blanks, use the RightTrim function. To remove leading blanks, use the LeftTrim function. To remove leading and trailing blanks, use the Trim function. For information about these functions, see RightTrim on page 856, LeftTrim on page 666, and Trim on page 1045.

Null value evaluations When you form a boolean expression that contains a null value, the AND and OR operators behave differently. Thinking of null as *undefined* (neither true nor false) makes the results easier to calculate.

For more information about null values, see "NULL values" on page 8.

Examples

Case-sensitive comparisons If you compare two strings with the same text but different case, the comparison fails. But if you use the Upper or Lower function, you can ensure that the case of both strings are the same so that only the content affects the comparison:

```
City1 = "Austin"
City2 = "AUSTIN"
IF City1 = City2 ...              // Returns FALSE

City1 = "Austin"
City2 = "AUSTIN"
IF Upper(City1) = Upper(City2)... // Returns TRUE
```

Trailing blanks in comparisons In this example, trailing blanks in one string cause the comparison to fail:

```
City1 = "Austin"
City2 = "Austin       "
IF City1 = City2 ...             // Returns FALSE
```

Logical expressions with null values In this example, the expressions involving the variable *f*, which has been set to null, have null values:

```
boolean d, e = TRUE, f
SetNull(f)
d = e and f   // d is NULL
d = e or f    // d is TRUE
```

Concatenation operator in PocketBuilder

Description

The PocketBuilder concatenation operator joins the contents of two variables of the same type to form a longer value. You can concatenate strings and blobs.

The following table shows the concatenation operator.

Table 4-3: PocketBuilder concatenation operator

Operator	Meaning	Example
+	Concatenate	`"cat " + "dog"`

Examples

Example 1 These examples concatenate several strings:

```
string Test
Test = "over" + "stock" // Test contains "overstock"
string Lname, Fname, FullName
FullName = Lname + ', ' + Fname
      // FullName contains last name and first name,
      // separated by a comma and space.
```

Example 2 This example shows how a blob can act as an accumulator when reading data from a file:

```
integer i, fnum, loops
bflob tot_b, b
. . .
FOR i = 1 to loops
 bytes_read = FileRead(fnum, b)
 tot_b = tot_b + b
NEXT
```

Operator precedence in PocketBuilder expressions

Order of precedence

To ensure predictable results, all operators in a PocketBuilder expression are evaluated in a specific order of precedence. When the operators have the same precedence, PocketBuilder evaluates them left to right.

These are the operators in descending order of precedence:

Table 4-4: Order of precedence of operators

Operator	Purpose
( )	Grouping (see note below on overriding)
+, -	Unary plus and unary minus (indicates positive or negative number)
^	Exponentiation
*, /	Multiplication and division
+, -	Addition and subtraction; string concatenation
=, >, <, <=, >=, <>	Relational operators
NOT	Negation
AND	Logical and
OR	Logical or

How to override

To override the order, enclose expressions in parentheses. This identifies the group and order in which PocketBuilder will evaluate the expressions. When there are nested groups, the groups are evaluated from the inside out.

For example, in the expression `(x+(y*(a+b)))`, `a+b` is evaluated first. The sum of *a* and *b* is then multiplied by *y*, and this product is added to *x*.

Datatype of PocketBuilder expressions

General information

The datatype of an expression is important when it is the argument for a function or event. The expression's datatype must be compatible with the argument's definition. If a function is overloaded, the datatype of the argument determines which version of the function to call.

There are three types: numeric, string, and char datatypes.

Numeric datatypes in PocketBuilder

General information

All numeric datatypes are compatible with each other.

What PocketBuilder does

PocketBuilder converts datatypes as needed to perform calculations and make assignments. When PocketBuilder evaluates a numeric expression, it converts the datatypes of operands to datatypes of higher precedence according to the operators and the datatypes of other values in the expression.

Datatype promotion when evaluating numeric expressions

Order of precedence

The PocketBuilder numeric datatypes are listed here in order of highest to lowest precedence (the order is based on the range of values for each datatype):

Double
Real
Decimal
LongLong
UnsignedLong
Long
UnsignedInteger
Integer

Rules for type promotion

Datatypes of operands If operands in an expression have different datatypes, the value whose type has lower precedence is converted to the datatype with higher precedence.

Unsigned versus signed Unsigned has precedence over signed, so if one operand is signed and the other is unsigned, both are promoted to the unsigned version of the higher type. For example, if one operator is a long and another UnsignedInteger, both are promoted to UnsignedLong.

Operators The effects of operators on an expression's datatype are:

- **+, -, *** The minimum precision for addition, subtraction, and multiplication calculations is long. Integer types are promoted to long types before doing the calculation and the expression's resulting datatype is, at a minimum, long. When operands have datatypes of higher precedence, other operands are promoted to match based on the *Datatypes of operands* rule above.
- **/ and ^** The minimum precision for division and exponentiation is double. All types are promoted to double before doing the calculation, and the expression's resulting datatype is double.
- **Relational** Relational operators do not cause promotion of numeric types.

Datatypes of literals

When a literal is an operand in an expression, its datatype is determined by the literal's value. The datatype of a literal affects the type promotion of the literal and other operands in an expression.

Table 4-5: Datatypes of literal operands in an expression

Literal	Datatype
Integer literals (no decimal point or exponent) within the range of Long	Long
Integer literals beyond the range of Long and within the range of UnsignedLong	UnsignedLong
Integer literals beyond the range of UnsignedLong and within the range of LongLong	UnsignedLong
Numeric literals with a decimal point (but no exponent)	Decimal
Numeric literals with a decimal point and explicit exponent	Double

Out of range
Integer literals beyond the range of LongLong cause compiler errors.

Assignment and datatypes

General information

Assignment is not part of expression evaluation. In an assignment statement, the value of an expression is converted to the datatype of the left-hand variable. In the expression

```
c = a + b
```

the datatype of `a+b` is determined by the datatypes of *a* and *b*. Then, the result is converted to the datatype of *c*.

Overflow on assignment

Even when PocketBuilder performs a calculation at high enough precision to handle the results, assignment to a lower precision variable can cause overflow, producing the wrong result.

Example 1 Consider this code:

```
integer a = 32000, b = 1000
long d
d = a + b
```

The final value of *d* is 33000. The calculation proceeds like this:

Convert integer *a* to long
Convert integer *b* to long
Add the longs *a* and *b*
Assign the result to the long *d*

Because the variable *d* is a long, the value 33000 does not cause overflow.

Example 2 In contrast, consider this code with an assignment to an integer variable:

```
integer a = 32000, b = 1000, c
long e
c = a + b
e = c
```

The resulting value of *c* and *e* is -32536. The calculation proceeds like this:

Convert integer *a* to long
Convert integer *b* to long
Add the longs *a* and *b*
Convert the result from long to integer and assign the result to *c*
Convert integer *c* to long and assign the result to *e*

The assignment to *c* causes the long result of the addition to be truncated, causing overflow and wrapping. Assigning *c* to *e* cannot restore the lost information.

String and char datatypes in PocketBuilder

General information

There is no explicit char literal type.

String literals convert to type char using the following rules:

- When a string literal is assigned to a char variable, the first character of the string literal is assigned to the variable. For example:

  ```
  char c = "xyz"
  ```

  results in the character *x* being assigned to the char variable *c*.

- Special characters (such as newline, formfeed, octal, hex, and so on) can be assigned to char variables using string conversion, such as:

  ```
  char c = "~n"
  ```

String variables assigned to char variables also convert using these rules. A char variable assigned to a string variable results in a one-character string.

Assigning strings to char arrays

As with other datatypes, you can use arrays of chars. Assigning strings to char arrays follows these rules:

- If the char array is unbounded (defined as a variable-size array), the contents of the string are copied directly into the char array.
- If the char array is bounded and its length is less than or equal to the length of the string, the string is truncated in the array.
- If the char array is bounded and its length is greater than the length of the string, the entire string is copied into the array along with its zero terminator. Remaining characters in the array are undetermined.

Assigning char arrays to strings

When a char array is assigned to a string variable, the contents of the array are copied into the string up to a zero terminator, if found, in the char array.

Using both strings and chars in an expression

Expressions using both strings and char arrays promote the chars to strings before evaluation. For example, the following promotes the contents of *c* to a string before comparison with the string “x”:

```
char c
. . .
if (c = "x") then
```

Using chars in PowerScript functions

All PowerScript functions that take strings also take chars and char arrays, subject to the conversion rules described above.

CHAPTER 5 Structures and Objects

About this chapter

This chapter describes basic concepts for structures and objects and how you define, declare, and use them in PowerScript.

Contents

About structures

General information

A structure is a collection of one or more variables (sometimes called elements) that you want to group together under a single name. The variables can have any datatype, including standard and object datatypes and other structures.

Defining structures

When you define a structure in the Structure painter or an object painter (such as Window, Menu, or User Object), you are creating a structure definition. To use the structure, you must declare it. When you declare it, an instance of it is automatically created for you. When it goes out of scope, the structure is destroyed.

For details about defining structures, see the *Users Guide*.

Declaring structures

If you have defined a global structure in the Structure painter called str_emp_data, you can declare an instance of the structure in a script or in an object's instance variables. If you define the structure in an object painter, you can only declare instances of the structure in the object's instance variables and scripts.

This declaration declares two instances of the structure str_emp_data:

```
str_emp_data str_emp1, str_emp2
```

Referring to structure variables

In scripts, you refer to the structure's variables using dot notation:

structurename.*variable*

These statements assign values to the variables in str_emp_data:

```
str_emp1.emp_id = 100
str_emp1.emp_lname = "Jones"
str_emp1.emp_salary = 200

str_emp2.emp_id = 101
str_emp2.emp_salary = str_emp1.salary * 1.05
```

Using structures as instance variables

If the structure is declared as part of an object, you can qualify the structure name using dot notation:

objectname.*structurename*.*variable*

Suppose that this declaration is an instance variable of the window w_customer:

```
str_cust_data str_cust1
```

The following statement in a script for the object refers to a variable of str_cust_data. The pronoun This is optional, because the structure declaration is part of the object:

```
This.str_cust1.name
```

The following statement in a script for some other object qualifies the structure with the window name:

```
w_customer.str_cust1.name
```

About objects

What an object is

In object-oriented programming, an object is a self-contained module containing state information and associated methods. Most entities in PocketBuilder are objects: visual objects such as windows and controls on windows, nonvisual objects such as transaction and error objects, and user objects that you design yourself.

An object class is a definition of an object. You create an object's definition in the appropriate painter: Window, Menu, Application, Structure, or User Object painter. In the painter, you add controls to be part of the object, specify initial values for the object's properties, define its instance variables and functions, and write scripts for its events and functions.

An object instance is an occurrence of the object created during the execution of your application. Your code instantiates an object when it allocates memory for the object and defines the object based on the definition in the object class.

An object reference is your handle to the object instance. To interact with an object, you need its object reference. You can assign an object reference to a variable of the appropriate type.

System objects versus user objects

There are two categories of objects supported by PocketBuilder: system objects (also referred to as system classes) defined by PocketBuilder and user objects you in define in painters.

System objects The PocketBuilder system objects or classes are inherited from the base class PowerObject. The system classes are the ancestors of all the objects you define. To see the system class hierarchy, select the System tab in the Browser, select PowerObject, and select Show Hierarchy and Expand All from the pop-up menu.

User objects You can create user object class definitions in several painters: Window, Menu, Application, Structure, and User Object painters. The objects you define are inherited from one of the system classes or another of your classes.

Some painters use many classes. In the Window and User Object painters, the main definition is inherited from the window or user object class. The controls you use are also inherited from the system class for that control.

About user objects

Two types

There are two major types of user objects: visual and class.

Visual user objects

A visual user object is a reusable control or set of controls that has a certain behavior. There are three types—standard, custom, and external.

Table 5-1: Visual user object types

Visual user objects	Description
Standard	Inherited from a specific visual control. You can set properties and write scripts so that the control is ready for use. It has the same events and properties as the control it is inherited from plus any that you add.

Visual user objects	Description
Custom	Inherited from the UserObject system class. You can include many controls in the user object and write scripts for their events. Each control in the user object has the same events and properties as the controls from which they are inherited plus any that you add.
External	A user object that displays a visual control defined in a DLL. The control is not part of the PocketBuilder object hierarchy. The DLL developer provides information for setting style bits that control its presentation. Its events, functions, and properties are specified by the developer of the DLL.

Class user objects

Class user objects consist of properties, functions, and sometimes events. They have no visual component. There are two types—standard and custom.

Table 5-2: Class user object types

Class user objects	Description
Standard	Inherits its definition from a nonvisual PocketBuilder object, such as the Transaction or Error object. You can add instance variables and functions. A few nonvisual objects have events—to write scripts for these events, you have to define a class user object.
Custom	An object of your own design for which you define instance variables, events, and functions in order to encapsulate application-specific programming in an object.

For information on defining and using user objects, see the *Users Guide*.

Instantiating objects

Classes versus instances

Because of the way PocketBuilder object classes and instances are named, it is easy to think they are the same thing. For example, when you define a window in the Window painter, you are defining an object class.

One instance

When you open a window with the simplest format of the Open function, you are instantiating an object instance. Both the class definition and the instance have the same name. In your application, *w_main* is a global variable of type w_main:

```
Open(w_main)
```

When you open a window this way, you can only open one instance of the object.

Several instances

If you want to open more than one instance of a window class, you need to define a variable to hold each object reference:

```
w_main w_1, w_2
Open(w_1)
Open(w_2)
```

You can also open windows by specifying the class in the Open function:

```
window w_1, w_2
Open(w_1, "w_main")
Open(w_2, "w_main")
```

For class user objects, you always define a variable to hold the object reference and then instantiate the object with the CREATE statement:

```
uo_emp_data uo_1, uo_2
uo_1 = CREATE uo_emp_data
uo_2 = CREATE uo_emp_data
```

You can have more than one reference to an object. You might assign an object reference to a variable of the appropriate type, or you might pass an object reference to another object so that it can change or get information from the object.

For more information about object variables and assignment, see “User objects that behave like structures” on page 79.

Using ancestors and descendants

Descendent objects

In PocketBuilder, an object class can be inherited from another class. The inherited or descendent object has all the instance variables, events, and functions of the ancestor. You can augment the descendant by adding more variables, events, and functions. If you change the ancestor, even after editing the descendant, the descendant incorporates the changes.

Instantiating

When you instantiate a descendent object, PocketBuilder also instantiates all its ancestor classes. You do not have programmatic access to these ancestor instances, except in a few limited ways, such as when you use the scope operator to access an ancestor version of a function or event script.

Garbage collection

What garbage collection does

The PocketBuilder garbage collection mechanism checks memory automatically for unreferenced and orphaned objects and removes any it finds, thus taking care of most memory leaks. You can use garbage collection to destroy objects instead of explicitly destroying them using the DESTROY statement. This lets you avoid execution-time errors that occur when you destroy an object that was being used by another process or had been passed by reference to a posted event or function.

When garbage collection occurs

Garbage collection occurs:

- **When a reference is removed from an object** A reference to an object is any variable whose value is the object. When the variable goes out of scope, or when it is assigned a different value, PocketBuilder removes a reference to the object, counts the remaining references, and destroys the object if no references remain.
- **When the garbage collection interval is exceeded** When PocketBuilder completes the execution of a system-triggered event, it makes a garbage collection pass if the set interval between garbage collection passes has been exceeded. The default interval is 0.5 seconds. The garbage collection pass removes any objects and classes that cannot be referenced, including those containing circular references (otherwise unreferenced objects that reference each other).

Posting events and functions
When you post an event or function and pass an object reference, PocketBuilder adds an internal reference to the object to prevent it from being collected between the time of the post and the actual execution of the event or function. This reference is removed when the event or function is executed.

Exceptions to garbage collection

There are a few objects that are prevented from being collected:

- **Visual objects** Any object that is visible on your screen is not collected because when the object is created and displayed on your screen, an internal reference is added to the object. When any visual object is closed it is explicitly destroyed.
- **Timing objects** Any Timing object that is currently running is not collected because the Start function for a Timing object adds an internal reference. The Stop function removes the reference.
- **Shared objects** Registered shared objects are not collected because the SharedObjectRegister function adds an internal reference. SharedObjectUnregister removes the internal reference.

Controlling when garbage collection occurs

Garbage collection occurs automatically in PocketBuilder, but you can use the functions GarbageCollect, GarbageCollectGetTimeLimit, and GarbageCollectSetTimeLimit to force immediate garbage collection or to change the interval between reference count checks. By setting the interval between garbage collection passes to a very large number, you can effectively turn off garbage collection.

User objects that behave like structures

In PocketBuilder, a nonvisual user object can provide functionality similar to that of a structure. Its instance variables form a collection similar to the variables for the structure. In scripts, you use dot notation to refer to the user object's instance variables, just as you do for structure variables.

Advantages of user objects

The user object can include functions and its own structure definitions, and it allows you to inherit from an ancestor class. None of this is possible with a structure definition.

Memory allocation differences

Memory allocation is different for user objects and structures. An object variable is a reference to the object. Declaring the variable does not allocate memory for the object. After you declare it, you must instantiate it with a CREATE statement. Assignment for a user object is also different (described in "Assignment for objects and structures" next).

Autoinstantiated objects

If you want a user object that has methods and inheritance but want the memory allocation of a structure, you can define an autoinstantiated object.

You do not have to create and destroy autoinstantiated objects. Like structures, they are created when they are declared and destroyed when they go out of scope. However, because assignment for autoinstantiated objects behaves like structures, the copies made of the object can be a drawback.

To make a custom class user object autoinstantiated, select the Autoinstantiate check box on the user object's property sheet.

Assignment for objects and structures

In PocketBuilder, assignment for objects is different from assignment for structures or autoinstantiated objects:

- When you assign one structure to another, the whole structure is copied so that there are two copies of the structure.
- When you assign one object variable to another, the object reference is copied so that both variables point to the same object. There is only one copy of the object.

Assignment for structures

Declaring a structure variable creates an instance of that structure:

```
str_emp_data str_emp1, str_emp2 // Two structure
                                   // instances
```

When you assign a structure to another structure, the whole structure is copied and a second copy of the structure data exists:

```
str_emp1 = str_emp2
```

The assignment copies the whole structure from one structure variable to the other. Each variable is a separate instance of the structure str_emp_data.

Restriction on assignment

If the structures have different definitions, you cannot assign one to another, even if they have the same set of variable definitions.

For example, this assignment is not allowed:

```
str_emp str_person1
str_cust str_person2
str_person2 = str_person1 // Not allowed
```

For information about passing structures as function arguments, see “Passing arguments to functions and events” on page 99.

Assignment for objects

Declaring an object variable declares an object reference:

```
uo_emp_data uo_emp1, uo_emp2 // Two object references
```

Using the CREATE statement creates an instance of the object:

```
uo_emp1 = CREATE uo_emp_data
```

When you assign one object variable to another, a reference to the object instance is copied. Only one copy of the object exists:

```
uo_emp2 = uo_emp1 // Both point to same object instance
```

Ancestor and descendent objects

Assignments between ancestor and descendent objects occur in the same way, with an object reference being copied to the target object.

Suppose that uo_emp_data is an ancestor user object of uo_emp_active and uo_emp_inactive.

Declare variables of the ancestor type:

```
uo_emp_data uo_emp1, uo_emp2
```

Create an instance of the descendant and store the reference in the ancestor variable:

```
uo_emp1 = CREATE USING "uo_emp_active"
```

Assigning *uo_emp1* to *uo_emp2* makes both variables refer to one object that is an instance of the descendant uo_emp_active:

```
uo_emp2 = uo_emp1
```

For information about passing objects as function arguments, see “Passing arguments to functions and events” on page 99.

Assignment for autoinstantiated user objects

Declaring an autoinstantiated user object creates an instance of that object (just like a structure). The CREATE statement is not allowed for objects with the Autoinstantiate setting. In the following example, uo_emp_data has the Autoinstantiate setting:

```
uo_emp_data uo_emp1, uo_emp2 // Two object instances
```

When you assign an autoinstantiated object to another autoinstantiated object, the *whole object* is copied to the second variable:

```
uo_emp1 = uo_emp2
```

You never have multiple references to an autoinstantiated user object.

Passing to a function

When you pass an autoinstantiated user object to a function, it behaves like a structure:

- Passing by value passes a copy of the object.
- Passing by reference passes a pointer to the object variable, just as for any standard datatype.
- Passing as read-only passes a copy of the object but that copy cannot be modified.

Restrictions for copying

Assignments are allowed between autoinstantiated user objects only if the object types match or if the target is a nonautoinstantiated ancestor.

Rule 1 If you assign one autoinstantiated object to another, they must be of the same type.

Rule 2 If you assign an autoinstantiated descendent object to an ancestor variable, the ancestor *cannot* have the Autoinstantiate setting. The ancestor variable will contain a reference to a copy of its descendant.

Rule 3 If you assign an ancestor object to a descendent variable, the ancestor must contain an instance of the descendant or an execution error occurs.

Examples

To illustrate, suppose you have these declarations. Uo_emp_active and uo_emp_inactive are autoinstantiated objects that are descendants of non-autoinstantiated uo_emp_data:

```
uo_emp_data uo_emp1 // Ancestor
uo_emp_active uo_empa, uo_empb // Descendants
uo_emp_inactive uo_empi // Another descendant
```

Example of rule 1 When assigning one instance to another from the user objects declared above, some assignments are not allowed by the compiler:

```
uo_empb = uo_empa // Allowed, same type
uo_empa = uo_empi // Not allowed, different types
```

Example of rule 2 After this assignment, *uo_emp1* contains a copy of the descendent object *uo_empa*. Uo_emp_data (the type for *uo_emp1*) must not be autoinstantiated. Otherwise, the assignment violates rule 1. If *uo_emp1* is autoinstantiated, a compiler error occurs:

```
uo_emp1 = uo_empa
```

Example of rule 3 This assignment is only allowed if *uo_emp1* contains an instance of its descendant *uo_empa*, which it would if the previous assignment had occurred before this one:

```
uo_empa = uo_emp1
```

If it did not contain an instance of target descendent type, an execution error would occur.

For more information about passing arguments to functions and events, see “Passing arguments to functions and events” on page 99.

CHAPTER 6 Calling Functions and Events

About this chapter

This chapter provides background information that will help you understand the different ways you can use functions and events. It then provides the syntax for calling functions and events.

Contents

About functions and events

Importance of functions and events

Much of the power of the PowerScript language resides in the built-in PowerScript functions that you can use in expressions and assignment statements.

Types of functions and events

PocketBuilder objects have built-in events and functions. You can enhance objects with your own user-defined functions and events, and you can declare local external functions for an object. The PowerScript language also has system functions that are not associated with any object. You can define your own global functions and declare external functions and remote procedure calls.

The following table shows the different types of functions and events.

Table 6-1: Types of functions and events

Category	Item	Definition
Events	Event	An action in an object or control that can start the execution of a script. A user can initiate an event by an action such as clicking an object or entering data, or a statement in another script can initiate the event.
	User event	An event you define to add functionality to an object. You specify the arguments, return value, and whether the event is mapped to a system message. For information about defining user events, see the *Users Guide*.
	System or built-in event	An event that is part of an object's PocketBuilder definition. System events are usually triggered by user actions or system messages. PocketBuilder passes a predefined set of arguments for use in the event's script. System events either return a long or do not have a return value.
Functions	Function	A program or routine that performs specific processing.
	System function	A built-in PowerScript function that is not associated with an object.
	Object function	A function that is part of an object's definition. PocketBuilder has many predefined object functions and you can define your own.
	User-defined function	A function you define. You define global functions in the Function painter and object functions in other painters with Script views.
	Global function	A function you define that can be called from any script. PowerScript's system functions are globally accessible, but they have a different place in the search order.
	Local external function	An external function that belongs to an object. You declare it in the Window or User Object painter. Its definition is in another library.
	Global external function	An external function that you declare in any painter, making it globally accessible. Its definition is in another library.
	Remote procedure call (RPC)	A stored procedure in a database that you can call from a script. The declaration for an RPC can be global or local (belonging to an object). The definition for the procedure is in the database.

Comparing functions and events

Functions and events have the following similarities:

- Both functions and events have arguments and return values.
- You can call object functions and events dynamically or statically. Global or system functions cannot be called dynamically.
- You can post or trigger a function or event call.

Functions and events have the following differences:

- Functions can be global or part of an object's definition. Events are associated only with objects.
- PocketBuilder uses different search orders when looking for events and functions.
- A call to an undefined function triggers an error. A call to an undefined event does not trigger an error.
- Object-level functions can be overloaded. Events (and global functions) cannot be overloaded.
- When you define a function, you can restrict access to it. You cannot add scope restrictions when you define events.
- When functions are inherited, you can extend the ancestor function by calling it in the descendant's script. You can also override the function definition. When events are inherited, the scripts for those events are extended by default. You can choose to extend or override the script.

Which to use

Whether you write most of your code in user-defined functions or in event scripts is one of the design decisions you must make. Because there is no performance difference, the decision is based on how you prefer to interact with PocketBuilder: whether you prefer the interface for defining user events or that for defining functions, how you want to handle errors, and whether your design includes overloading.

It is unlikely that you will use either events or functions exclusively, but for ease of maintenance, you might want to choose one approach for handling most situations.

Finding and executing functions and events

PocketBuilder looks for a matching function or event based on its name and its argument list. PocketBuilder can make a match between compatible datatypes (such as all the numeric types). The match does not have to be exact. PocketBuilder ranks compatible datatypes to quantify how closely one datatype matches another.

A major difference between functions and events is how PocketBuilder looks for them.

Finding functions

When calling a function, PocketBuilder searches until it finds a matching function and executes it—the search ends. Using functions with the same name but different arguments is called function overloading. For more information, see "Overloading, overriding, and extending functions and events" on page 97.

Unqualified function names

If you do not qualify a function name with an object, PocketBuilder searches for the function and executes the first one it finds that matches the name and arguments. It searches for a match in the following order:

1. A global external function.
2. A global function.
3. An object function and local external function. If the object is a descendant, PocketBuilder searches upward through the ancestor hierarchy to find a match for the function prototype.
4. A system function.

DataWindow expression functions
The functions that you use in the DataWindow painter in expressions for computed fields, filters, validation rules, and graphed data cannot be overridden. For example, if you create a global function called Today, it is used instead of the PowerScript system function Today, but it is *not* used instead of the DataWindow expression function Today.

Qualified function names

You can qualify an object function using dot notation to ensure that the object function is found, not a global function of the same name. With a qualified name, the search for a matching function involves the ancestor hierarchy only (item 3 in the search list above), as shown in the following examples of function calls:

```
dw_1.Update( )
w_employee.uf_process_list()
This.uf_process_list()
```

When PocketBuilder searches the ancestor hierarchy for a function, you can specify that you want to call an ancestor function instead of a matching descendent function.

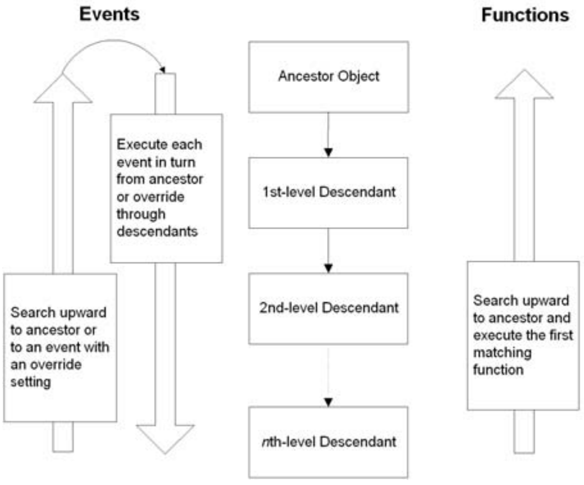
For the syntax for calling ancestor functions, see “Calling functions and events in an object’s ancestor” on page 108.

Finding events

PocketBuilder events in descendent objects are, by default, extensions of ancestor events. PocketBuilder searches for events in the object’s ancestor hierarchy until it gets to the top ancestor or finds an event that overrides its ancestor. Then it begins executing the events, from the ancestor event down to the descendent event.

Finding functions versus events

The following illustration shows the difference between searching for events and searching for functions:


Events
Functions
Ancestor Object
Execute each event in turn from ancestor or override through descendants
1st-level Descendant
2nd-level Descendant
Search upward to ancestor or to an event with an override setting
Search upward to ancestor and execute the first matching function
nth-level Descendant

Triggering versus posting functions and events

Triggering

In PocketBuilder, when you trigger a function or event, it is called immediately. Its return value is available for use in the script.

Posting

When you post a function or event, it is added to the object's queue and executed in its turn. In most cases, it is executed when the current script is finished; however, if other system events have occurred in the meantime, its position in the queue might be after other scripts. Its return value is not available to the calling script.

Because POST makes the return value unavailable to the caller, you can think of it as turning the function or event call into a statement.

Use posting when activities need to be finished before the code checks state information or does further processing (see Example 2 below).

PocketBuilder messages processed first

All events posted by PocketBuilder are processed by a separate queue from the Windows system queue. PocketBuilder posted messages are processed before Windows posted messages, so PocketBuilder events that are posted in an event that posts a Windows message are processed before the Windows message.

For example, when a character is typed into an EditMask control, the PocketBuilder pdm_keydown event posts the Windows message WM_CHAR to enter the character. If you want to copy the characters as they are entered from the EditMask control to another control, do not place the code in an event posted in the pdm_keydown event. The processing must take place in an event that occurs after the WM_CHAR message is processed, such as in an event mapped to pdm_keyup.

Restrictions for *POST*

Because no value is returned, you:

- Cannot use a posted function or event as an operand in an expression
- Cannot use a posted function or event as the argument for another function
- Can only use POST on the last call in a cascaded sequence of calls

These statements cause a compiler error. Both uses require a return value:

```
IF POST IsNull( ) THEN ...
w_1.uf_getresult(dw_1.POST GetBorderStyle(2))
```

***TriggerEvent* and *PostEvent* functions**
For backward compatibility, the TriggerEvent and PostEvent functions are still available, but you cannot pass arguments to the called event. You must pass data to the event in PocketBuilder's Message object.

Examples of posting

The following examples illustrate how to post events.

Example 1 In a sample application, the Open event of the w_activity_manager window calls the functions uf_setup and uf_set_tabpgsystem. (The functions belong to the user object u_app_actman.) Because the functions are posted, the Open event is allowed to finish before the functions are called. The result is that the window is visible while setup processing takes place, giving the user something to look at:

```
guo_global_vars.iuo_app_actman.POST uf_setup()
guo_global_vars.iuo_com_actman.POST
uf_set_tabpgsystem(0)
```

Example 2 In a sample application, the DoubleClicked event of the tv_roadmap TreeView control in the u_tabpg_amroadmap user object posts a function that processes the TreeView item. If the event is not posted, the code that checks whether to change the item's picture runs before the item's expanded flag is set:

```
parent.POST uf_process_item ()
```

Static versus dynamic calls

Calling functions and events

PocketBuilder calls functions and events in three ways, depending on the type of function or event and the lookup method defined.

Table 6-2: How PocketBuilder calls functions and events

Type of function	Compiler typing	Comments
Global and system functions	Strongly typed. The function *must* exist when the script is compiled.	These functions must exist and are called directly. They are not polymorphic, and no substitution is ever made at execution time.
Object functions with STATIC lookup	Strongly typed. The function *must* exist when the script is compiled.	The functions are polymorphic. They must exist when you compile, but if another class is instantiated at execution time, its function is called instead.
Object functions with DYNAMIC lookup	Weakly typed. The function does *not* have to exist when the script is compiled.	The functions are polymorphic. The actual function called is determined at execution time.

Specifying static or dynamic lookup

For object functions and events, you can choose when PocketBuilder looks for them by specifying static or dynamic lookup. You specify static or dynamic lookup using the STATIC or DYNAMIC keywords. The DYNAMIC keyword applies only to functions that are associated with an object. You cannot call global or system functions dynamically.

Static calls

By default, PocketBuilder makes static lookups for functions and events. This means that it identifies the function or event by matching the name and argument types when it compiles the code. A matching function or event must exist in the object at compile time.

Results of static calls

Static calls do not guarantee that the function or event identified at compile time is the one that is executed. Suppose that you define a variable of an ancestor type and it has a particular function definition. If you assign an instance of a descendent object to the variable and the descendant has a function that overrides the ancestor's function (the one found at compile time), the function in the descendant is executed.

Dynamic calls

When you specify a dynamic call in PocketBuilder, the function or event does not have to exist when you compile the code. You are indicating to the compiler that there will be a suitable function or event available at execution time.

For a dynamic call, PocketBuilder waits until it is time to execute the function or event to look for it. This gives you flexibility and allows you to call functions or events in descendants that do not exist in the ancestor.

Results of dynamic calls

To illustrate the results of dynamic calls, consider these objects:

- Ancestor window w_a with a function Set(*integer*).
- Descendent window w_a_desc with two functions: Set(*integer*) overrides the ancestor function, and Set(*string*) is an overload of the function.

Situation 1 Suppose you open the window *mywindow* of the ancestor window class w_a:

```
w_a mywindow
Open(mywindow)
```

This is what happens when you call the Set function statically or dynamically:

This statement	Has this result
`mywindow.Set(1)`	Compiles correctly because function is found in the ancestor w_a. At runtime, Set(*integer*) in the *ancestor* is executed.
`mywindow.Set("hello")`	Fails to compile; no function prototype in w_a matches the call.
`mywindow.DYNAMIC Set("hello")`	Compiles successfully because of the DYNAMIC keyword. An error occurs at runtime because no matching function is found.

Situation 2 Now suppose you open *mywindow* as the descendant window class w_a_desc:

```
w_a mywindow
Open(mywindow, "w_a_desc")
```

This is what happens when you call the Set function statically or dynamically in the descendant window class:

This statement	Has this result
`mywindow.Set(1)`	Compiles correctly because function is found in the ancestor w_a. At runtime, Set(*integer*) in the *descendant* is executed.
`mywindow.Set("hello")`	Fails to compile; no function prototype in the ancestor matches the call.
`mywindow.DYNAMIC Set("hello")`	Compiles successfully because of the DYNAMIC keyword. At runtime, Set(*string*) in the *descendant* is executed.

Disadvantages of dynamic calls

Slower performance Because dynamic calls are resolved at runtime, they are slower than static calls. If you need the fastest performance, design your application to avoid dynamic calls.

Less error checking When you use dynamic calls, you are foregoing error checking provided by the compiler. Your application is more open to application errors, because functions that are called dynamically might be unavailable at execution time. Do not use a dynamic call when a static call will suffice.

Example using dynamic call

A sample application has an ancestor window w_datareview_frame that defines several functions called by the menu items of m_datareview_framemenu. They are empty stubs with empty scripts so that static calls to the functions will compile. Other windows that are descendants of w_datareview_frame have scripts for these functions, overriding the ancestor version.

The wf_print function is one of these—it has an empty script in the ancestor and appropriate code in each descendent window:

```
guo_global_vars.ish_currentsheet.wf_print ()
```

The wf_export function called by the m_export item on the m_file menu does not have a stubbed-out version in the ancestor window. This code for m_export uses the DYNAMIC keyword to call wf_export. When the program runs, the value of variable *ish_currentsheet* is a descendent window that does have a definition for wf_export:

```
guo_global_vars.ish_currentsheet.DYNAMIC wf_export()
```

Errors when calling functions and events dynamically

If you call a function or event dynamically, different conditions create different results, from no effect to an execution error. The tables in this section illustrate this.

Functions

The rules for functions are similar to those for events, except functions must exist: if a function is not found, an error always occurs. Although events can exist without a script, if a function is defined it has to have code. Consider the following statements:

1. This statement calls a function without looking for a return value:

   ```
   object.DYNAMIC funcname( )
   ```

2. This statement looks for an integer return value:

   ```
   int li_int
   li_int = object.DYNAMIC funcname( )
   ```

3. This statement looks for an Any return value:

   ```
   any la_any
   la_any = object.DYNAMIC funcname( )
   ```

The following table uses these statements as examples.

Table 6-3: Dynamic function calling errors

Condition 1	Condition 2	Result	Example
The function does not exist.	None.	Execution error 65: Dynamic function not found.	All the statements cause error 65.
The function is found and executed but is not defined with a return value.	The code is looking for a return value.	Execution error 63: Function/event with no return value used in expression.	Statements 2 and 3 cause error 63.

Events

Consider these statements:

1 This statement calls an event without looking for a return value:

```
object.EVENT DYNAMIC eventname( )
```

2 This example looks for an integer return value:

```
int li_int
li_int = object.EVENT DYNAMIC eventname( )
```

3 This example looks for an Any return value:

```
any la_any
la_any = object.EVENT DYNAMIC eventname( )
```

The following table uses these statements as examples.

Table 6-4: Dynamic event calling errors

Condition 1	Condition 2	Result	Example
The event does not exist.	The code *is not* looking for a return value.	Nothing; the call fails silently.	Statement 1 fails but does not cause an error.
	The code *is* looking for a return value.	A null of the Any datatype is returned.	*La_any* is set to null in statement 3.
		If the expected datatype is not Any, execution error 19 occurs: Cannot convert Any in Any variable to datatype.	The assignment to *li_int* causes execution error 19 in statement 2.

Condition 1	Condition 2	Result	Example
The event is found but is not implemented (there is no script).	The event *has* a defined return value.	A null of the defined datatype is returned.	If eventname is defined to return integer, *li_int* is set to null in statement 2.
	The event *does not have* a defined return value.	A null of the Any datatype is returned.	*La_any* is set to null in statement 3.
		If the expected datatype is not Any, execution error 19 occurs: Cannot convert Any in Any variable to datatype.	The assignment to *li_int* causes execution error 19 in statement 2.
The event is found and executed but is not defined with a return value.	The code is looking for a return value.	Execution error 63: Function/event with no return value used in expression.	Statements 2 and 3 cause error 63.

When an error occurs

You can surround a dynamic function call in a try-catch block to prevent the application from terminating when an execution error occurs. Although you can also handle the error in the SystemError event, you should not allow the application to continue once the SystemError event is invoked—the SystemError event should only clean up and halt the application.

For information on using try-catch blocks, see the section on exception handling in the *Resource Guide*.

If the arguments do not match

Function arguments are part of the function's definition. Therefore, if the arguments do not match (a compatible match, not an exact match), it is essentially a different function. The result is the same as if the function did not exist.

If you call an event dynamically and the arguments do not match, the call fails and control returns to the calling script. There is no error.

Error-proofing your code

Calling functions and events dynamically opens up your application to potential errors. The surest way to avoid these errors is to always make static calls to functions and events. When that is not possible, your design and testing can ensure that there is always an appropriate function or event with the correct return datatype.

One type of error you can check for and avoid is data conversion errors.

The preceding tables illustrated that a function or event can return a null value either as an Any variable or as a variable of the expected datatype when a function or event definition exists but is not implemented.

If you always assign return values to Any variables for dynamic calls, you can test for null (which indicates failure) before using the value in code.

This example illustrates the technique of checking for null before using the return value.

```
any la_any
integer li_gotvalue
la_any = object.DYNAMIC uf_getaninteger( )
IF IsNull(la_any) THEN
   ... // Error handling
ELSE
   li_gotvalue = la_any
END IF
```

Overloading, overriding, and extending functions and events

In PocketBuilder, when functions are inherited, you can choose to overload or override the function definition, described in "Overloading and overriding functions" next.

When events are inherited, the scripts for those events are extended by default. You can choose to extend or override the script, described in “Extending and overriding events” on page 99.

Overloading and overriding functions

To create an overloaded function, you declare the function as you would any function using Insert>Function.

Overriding means defining a function in a descendent object that has the same name and argument list as a function in the ancestor object. In the descendent object, the function in the descendant is always called instead of the one in the ancestor—unless you use the scope resolution operator (::).

To override a function, open the descendent object in the painter, select the function in the Script view, and code the new script. The icon that indicates that there is a script for a function is half shaded when the function is inherited from an ancestor.

You can overload or override object functions only—you cannot overload global functions.

Type promotion when matching arguments for overloaded functions

When you have overloaded a function so that one version handles numeric values and another version handles strings, it is clear to the programmer what arguments to provide to call each version of the function. Overloading with unrelated datatypes is a good idea and can provide needed functionality for your application.

Problematic overloading

If different versions of a function have arguments of related datatypes (different numeric types or strings and chars), you must consider how PocketBuilder promotes datatypes in determining which function is called. This kind of overloading is undesirable because of potential confusion in determining which function is called.

When you call a function with an *expression* as an argument, the datatype of the expression might not be obvious. However, the datatype is important in determining what version of an overloaded function is called.

Because of the intricacies of type promotion for numeric datatypes, you might decide that you should not define overloaded functions with different numeric datatypes. Changes someone makes later can affect the application more drastically than expected if the change causes a different function to be called.

How type promotion works

When PocketBuilder evaluates an expression, it converts the datatypes of constants and variables so that it can process or combine them correctly.

Numbers When PocketBuilder evaluates numeric expressions, it promotes the datatypes of values according to the operators and the datatypes of the other operands. For example, the datatype of the expression `n/2` is double because it involves division—the datatype of *n* does not matter.

Strings When evaluating an expression that involves chars and strings, PocketBuilder promotes chars to strings.

For more information on type promotion, see “Datatype of PocketBuilder expressions” on page 69.

Using conversion functions

You can take control over the datatypes of expressions by calling a conversion function. The conversion function ensures that the datatype of the expression matches the function prototype you want to call.

For example, because the expression `n/2` involves division, the datatype is `double`. However, if the function you want to call expects a `long`, you can use the `Long` function to ensure that the function call matches the prototype:

```
CalculateHalf(Long(n/2))
```

Extending and overriding events

In PocketBuilder, when you write event scripts in a descendent object, you can extend or override scripts that have been written in the ancestor.

Extending (the default) means executing the ancestor's script first, then executing code in the descendant's event script.

Overriding means ignoring the ancestor's script and only executing the script in the descendant.

No overloaded events
You cannot overload an event by defining an event with the same name but different arguments. Event names must be unique.

To select extending or overriding, open the script in the Script view and check or clear the Extend Ancestor Script item in the Edit or pop-up menu.

Passing arguments to functions and events

In PocketBuilder, arguments for built-in or user-defined functions and events can be passed three ways:

Table 6-5: Passing arguments to functions and events

Method of passing	Description
By value	A copy of the variable is available in the function or event script. Any changes to its value affect the copy only. The original variable in the calling script is not affected.

Method of passing	Description
By reference	A pointer to the variable is passed to the function or event script. Changes affect the original variable in the calling script.
Read-only	The variable is available in the function or event. Its value is treated as a constant—changes to the variable are not allowed and cause a compiler error. Read-only provides a performance advantage for some datatypes because it does not create a copy of the data, as with by value. Datatypes for which read-only provides a performance advantage are string, blob, date, time, and DateTime. For other datatypes, read-only provides documentation for other developers by indicating something about the purpose of the argument.

Passing objects

When you pass an object to a function or event, the object must exist when you refer to its properties and functions. If you call the function but the object has been destroyed, you get the execution error for a null object reference. This is true whether you pass by reference, by value, or read-only.

To illustrate, suppose you have a window with a SingleLineEdit. If you post a function in the window's Close event and pass the SingleLineEdit, the object does not exist when the function executes. To use information from the SingleLineEdit, you must pass the information itself, such as the object's text, rather than the object.

When passing an object, you never get another copy of the object. By reference and by value affect the object reference, not the object itself.

Objects passed by value

When you pass an object by value, you pass a copy of the reference to the object. That reference is still pointing to the original object. If you change properties of the object, you are changing the original object. However, you can change the value of the variable so that it points to another object without affecting the original variable.

Objects passed by reference

When you pass an object by reference, you pass a pointer to the original reference to the object. Again, if you change properties of the object, you are changing the original object. You can change the value of the variable that was passed, but the result is different—the original reference now points to the new object.

Objects passed as read-only

When you pass an object as read-only, you get a copy of the reference to the object. You cannot change the reference to point to a new object (because read-only is equivalent to a CONSTANT declaration), but you *can* change properties of the object.

Passing structures

Structures as arguments behave like simple variables, not like objects.

Structures passed by value

When you pass a structure by value, PocketBuilder passes a copy of the structure. You can modify the copy without affecting the original.

Structures passed by reference

When you pass a structure by reference, PocketBuilder passes a reference to the structure. When you changes values in the structure, you are modifying the original. You will not get a null object reference, because structures always exist until they go out of scope.

Structures passed as read-only

When you pass a structure as read-only, PocketBuilder passes a copy of the structure. You cannot modify any members of the structure.

Passing arrays

When an argument is an array, you specify brackets as part of the argument name in the declaration for the function or event.

Variable-size array as an argument

For example, suppose a function named uf_convertarray accepts a variable-size array of integers. If the argument's name is *intarray*, then for Name enter `intarray[ ]` and for Type enter `integer`.

In the script that calls the function, you either declare an array variable or use an instance variable or value that has been passed to you. The declaration of that variable, wherever it is, looks like this:

```
integer a[]
```

When you call the function, omit the brackets, because you are passing the whole array. If you specified brackets, you would be passing one value from the array:

```
uf_convertarray(a)
```

Fixed-size array as an argument

For comparison, suppose the uf_convertarray function accepts a fixed-size array of integers of 10 elements instead. If the argument's name is *intarray*, then for Name enter `intarray[10]`, and for Type enter `integer`.

The declaration of the variable to be passed looks like this:

```
integer a[10]
```

You call the function the same way, without brackets:

```
uf_convertarray(a)
```

If the array dimensions do not match
If the dimensions of the array variable passed do not match the dimensions declared for the array argument, then array-to-array assignment rules apply. For more information, see "Declaring arrays" on page 45.

Using return values

You can use return values of functions and events.

Functions

All built-in PowerScript functions return a value. You can use the return value or ignore it. User-defined functions and external functions might or might not return a value.

To use a return value, assign it to a variable of the appropriate datatype or call the function wherever you can use a value of that datatype.

Posting a function
If you post a function, you cannot use its return value.

Examples

The built-in Asc function takes a string as an argument and returns the ASCII value of the string's first character:

```
string S1 = "Carton"
int Test
Test=32+Asc(S1)   // Test now contains the value 99
                 // (the ASCII value of "C" is 67).
```

The SelectRow function expects a row number as the first argument. The return value of the GetRow function supplies the row number:

```
dw_1.SelectRow(dw_1.GetRow(), true)
```

To ignore a return value, call the function as a single statement:

```
Beep(4)       // This returns a value, but it is
               // rarely needed.
```

Events

Most system events return a value. The return value is a long—numeric codes have specific meanings for each event. You specify the event's return code with a RETURN statement in the event script.

When the event is triggered by user actions or system messages, the value is returned to the system, not to a script you write.

When you trigger a system or user-defined event, the return value is returned to your script and you can use the value as appropriate. If you post an event, you cannot use its return value.

Using cascaded calling and return values

PocketBuilder dot notation allows you to chain together several object function or event calls. The return value of the function or event becomes the object for the following call.

This syntax shows the relationship between the return values of three cascaded function calls:

func1returnsobject( ).func2returnsobject( ).func3returnsanything( )

Disadvantage of cascaded calls
When you call several functions in a cascade, you cannot check their return values and make sure they succeeded. If you want to check return values (and checking is always a good idea), call each function separately and assign the return values to variables. Then you can use the verified variables in dot notation before the final function name.

Dynamic calls

If you use the DYNAMIC keyword in a chain of cascaded calls, it carries over to all function calls that follow.

In this example, both func1 and func2 are called dynamically:

```
object1.DYNAMIC func1().func2()
```

The compiler reports an error if you use DYNAMIC more than once in a cascaded call. This example would cause an error:

```
object1.DYNAMIC func1().DYNAMIC func2() // error
```

Posted functions and events Posted functions and events do not return a value to the calling scripts. Therefore, you can only use POST for the last function or event in a cascaded call. Calls before the last must return a valid object that can be used by the following call.

System events System events can only be last in a cascaded list of calls, because their return value is a long (or they have no return value). They do not return an object that can be used by the next call.

An event you have defined can have a return value whose datatype is an object. You can include such events in a cascaded call.

Syntax for calling PocketBuilder functions and events

Description This syntax is used to call all PocketBuilder functions and events. Depending on the keywords used, this syntax can be used to call system, global, object, user-defined, and external functions as well as system and user-defined events.

Syntax

{ *objectname*.} { *type* } { *calltype* } { *when* } *name* ({ *argumentlist* })

The following table describes the arguments used in function and event calls.

Table 6-6: Arguments for calling functions and events

Argument	Description
objectname (optional)	The name of the object where the function or event is defined followed by a period or the descendant of that object/the name of the ancestor class followed by two colons. If a function name is not qualified, PocketBuilder uses the rules for finding functions and executes the first matching function it finds. For system or global functions, omit *objectname.* For the rules PocketBuilder uses to find unqualified function names, see "Finding and executing functions and events" on page 88.
type (optional)	A keyword specifying whether you are calling a function or event. Values are: • FUNCTION (Default) • EVENT

Argument	Description
calltype (optional)	A keyword specifying when PocketBuilder looks for the function or event. Values are: • STATIC (Default) • DYNAMIC For more information about static versus dynamic calls, see "Static versus dynamic calls" on page 91. For more information on dynamic calls, see "Dynamic calls" on page 92.
when (optional)	A keyword specifying whether the function or event should execute immediately or after the current script is finished. Values are: • TRIGGER — (Default) Execute it immediately. • POST — Put it in the object's queue and execute it in its turn, after other pending messages have been handled. For more about triggering and posting, see "Triggering versus posting functions and events" on page 90.
name	The name of the function or event you want to call.
argumentlist (optional)	The values you want to pass to *name*. Each value in the list must have a datatype that corresponds to the declared datatype in the function or event definition or declaration.

Usage

Case insensitivity
Function and event names are not case sensitive. For example, the following three statements are equivalent:

```
Clipboard("PocketBuilder")
clipboard("PocketBuilder")
CLIPBOARD("PocketBuilder")
```

Calling arguments The type, calltype, and when keywords can be in any order after *objectname*.

Not all options in the syntax apply to all types. For example, there is no point in calling a system PowerScript object function dynamically. It always exists, and the dynamic call incurs extra overhead. However, if you had a user-defined function of the same name that applied to a different object, you might call that function dynamically.

User-defined global functions and system functions can be triggered or posted but they cannot be called dynamically.

Finding functions If a global function does not exist with the given name, PocketBuilder will look for an object function that matches the name and argument list before it looks for a PocketBuilder system function.

Calling functions and events in the ancestor If you want to circumvent the usual search order and force PocketBuilder to find a function or event in an ancestor object, bypassing it in the descendant, use the ancestor operator (::).

For more information about the scope operator for ancestors, see “Calling functions and events in an object’s ancestor” on page 108.

Cascaded calls Calls can be cascaded using dot notation. Each function or event call must return an object type that is the appropriate object for the following call.

For more information about cascaded calls, see “Using cascaded calling and return values” on page 103.

Using return values If the function has a return value, you can call the function on the right side of an assignment statement, as an argument for another function, or as an operand in an expression.

External functions Before you can call an external function, you must declare it. For information about declaring external functions, see “Declaring external functions” on page 54.

Examples

Example 1 The following statements show various function calls using the most simple construction of the function call syntax.

This statement calls the system function Asc:

```
charnum = Asc("x")
```

This statement calls the DataWindow function in a script that belongs to the DataWindow:

```
Update( )
```

This statement calls the global user-defined function gf_setup_appl:

```
gf_setup_appl(24, "Window1")
```

This statement calls the system function PrintRect:

```
PrintRect(job, 250, 250, 7500, 1000, 50)
```

Example 2 The following statements show calls to global and system functions.

This statement posts the global user-defined function gf_setup_appl. The function is executed when the calling script finishes:

```
POST gf_setup_appl(24, "Window1")
```

This statement posts the system function PrintRect. It is executed when the calling script finishes. The print job specified in job must still be open:

```
POST PrintRect(job, 250, 250, 7500, 1000, 50)
```

Example 3 In a script for a control, these statements call a user-defined function defined in the parent window. The statements are equivalent, because FUNCTION, STATIC, and TRIGGER are the defaults:

```
Parent.FUNCTION STATIC TRIGGER wf_process( )
Parent.wf_process( )
```

Example 4 This statement in a DataWindow control's Clicked script calls the DoubleClicked event for the same control. The arguments the system passed to Clicked are passed on to DoubleClicked. When triggered by the system, PocketBuilder passes DoubleClicked those same arguments:

```
This.EVENT DoubleClicked(xpos, ypos, row, dwo)
```

This statement posts the same event:

```
This.EVENT POST DoubleClicked(xpos, ypos, row, dwo)
```

Windows CE platforms
Double-clicking is not a natural user action on Pocket PC devices, but it can be triggered if called in code or by quickly double-tapping an item with a stylus.

Example 5 The variable *iw_a* is an instance variable of an ancestor window type w_ancestorsheet:

```
w_ancestorsheet iw_a
```

A menu has a script that calls the wf_export function, but that function is not defined in the ancestor. The DYNAMIC keyword is required so that the script compiles:

```
iw_a.DYNAMIC wf_export( )
```

At execution time, the window that is opened is a descendant with a definition of wf_export. That window is assigned to the variable *iw_a* and the call to wf_export succeeds.

Calling functions and events in an object’s ancestor

Description

In PocketBuilder, when an object is instantiated with a descendant object, even if its class is the ancestor and that descendant has a function or event script that overrides the ancestor’s, the descendant’s version is the one that is executed. If you specifically want to execute the ancestor’s version of a function or event, you can use the ancestor operator (::) to call the ancestor’s version explicitly.

Syntax

{ *objectname*. } *ancestorclass* ::{ *type* } { *when* } *name* ({ *argumentlist* })

The following table describes the arguments used to call functions and events in an object’s ancestor.

Table 6-7: Arguments for calling ancestor functions and events

Argument	**Description**
objectname (optional)	The name of the object whose ancestor contains the function you want to execute.
ancestorclass	The name of the ancestor class whose function or event you want to execute. The pronoun Super provides the appropriate reference when *ancestorobject* is the immediate ancestor of the current object.
type (optional)	A keyword specifying whether you are calling a function or event. Values are: • (Default) FUNCTION • EVENT
when (optional)	A keyword specifying whether the function or event should execute immediately or after the current script is finished. Values are: • TRIGGER — (Default) Execute it immediately • POST — Put it in the object’s queue and execute it in its turn, after other pending messages have been handled
name	The name of the object function or event you want to call.
argumentlist (optional)	The values you want to pass to *name*. Each value in the list must have a datatype that corresponds to the declared datatype in the function definition.

Usage

The AncestorReturnValue variable When you extend an event script in a descendent object, the compiler automatically generates a local variable called AncestorReturnValue that you can use if you need to know the return value of the ancestor event script. The variable is also generated if you override the ancestor script and use the CALL syntax to call the ancestor event script.

The datatype of the AncestorReturnValue variable is always the same as the datatype defined for the return value of the event. The arguments passed to the call come from the arguments that are passed to the event in the descendent object.

Extending event scripts The AncestorReturnValue variable is always available in extended event scripts. When you extend an event script, PocketBuilder generates the following syntax and inserts it at the beginning of the event script:

CALL SUPER::*event_name*

You only see the statement if you export the syntax of the object or look at it in the Source editor.

The following example illustrates the code you can put in an extended event script:

```
If AncestorReturnValue = 1 THEN
// execute some code
ELSE
// execute some other code
END IF
```

Overriding event scripts The AncestorReturnValue variable is only available when you override an event script after you call the ancestor event using the CALL syntax:

CALL SUPER::*event_name*

or

CALL *ancestor_name*::*event_name*

The compiler cannot differentiate between the keyword SUPER and the name of the ancestor. The keyword is replaced with the name of the ancestor before the script is compiled.

The AncestorReturnValue variable is only declared and a value assigned when you use the CALL event syntax. It is not declared if you use the new event syntax:

ancestor_name::EVENT *event_name*( )

You can use the same code in a script that overrides its ancestor event script, but you must insert a CALL statement before you use the AncestorReturnValue variable.

```
// execute code that does some preliminary processing
CALL SUPER::uo_myevent
```

```
IF AncestorReturnValue = 1 THEN
...
```

For information about CALL, see CALL on page 116.

Examples

Example 1 Suppose a window w_ancestor has an event ue_process. A descendent window has a script for the same event.

This statement in a script in the descendant searches the event chain and calls all appropriate events. If the descendant extends the ancestor script, it calls a script for each ancestor in turn followed by the descendent script. If the descendant overrides the ancestor, it calls the descendent script only:

```
EVENT ue_process( )
```

This statement calls the ancestor event only (this script works if the calling script belongs to another object or the descendent window):

```
w_ancestor::EVENT ue_process( )
```

Example 2 You can use the pronoun Super to refer to the ancestor. This statement in a descendent window script or in a script for a control on that window calls the Clicked script in the immediate ancestor of that window.

```
Super::EVENT Clicked(0, x, y)
```

Example 3 These statements call a function wf_myfunc in the ancestor window (presumably, the descendant also has a function called wf_myfunc):

```
Super::wf_myfunc( )
Super::POST wf_myfunc( )
```

PART 2

Statements, Events, and Functions

CHAPTER 7

PowerScript Statements

About this chapter

This chapter describes the PowerScript statements and how to use them in scripts.

Contents

Assignment

Description

Assigns values to variables or object properties or object references to object variables.

Syntax

variablename = expression

Argument	Description
variablename	The name of the variable or object property to which you want to assign a value. *Variablename* can include dot notation to qualify the variable with one or more object names.
expression	An expression whose datatype is compatible with *variablename*.

Usage

Use assignment statements to assign values to variables. To assign a value to a variable anywhere in a script, use the equal sign (=). For example:

```
String1 = "Part is out of stock"
TaxRate = .05
```

No multiple assignments Since the equal sign is also a logical operator, you cannot assign more than one variable in a single statement. For example, the following statement does not assign the value 0 to A and B:

```
A=B=0    // This will not assign 0 to A and B.
```

This statement first evaluates `B=0` to true or FALSE and then tries to assign this boolean value to A. When A is not a boolean variable, this line produces an error when compiled.

Assigning array values You can assign multiple array values with one statement, such as:

```
int Arr[]
Arr = {1, 2, 3, 4}
```

You can also copy array contents. For example, this statement copies the contents of *Arr2* into array *Arr1*:

```
Arr1 = Arr2
```

Operator shortcuts The PowerScript shortcuts for assigning values to variables in the following table have slight performance advantages over their equivalents.

Table 7-1: Shortcuts for assigning values

Assignment	Example	Equivalent to
++	i ++	i = i + 1
--	i --	i = i - 1
+=	i += 3	i = i + 3
-=	i -= 3	i = i -3
*=	i *= 3	i = i * 3
/=	i /= 3	i = i / 3
^=	i ^=3	i = i ^ 3

Unless you have prohibited the use of dashes in variable names, you must leave a space before `--` and `-=`. If you do not, PowerScript reads the minus sign as part of a variable name. For more information, see “Identifier names” on page 5.

Examples

Example 1 These statements each assign a value to the variable *ld_date*:

```
date ld_date
ld_date = Today( )
ld_date = 1996-01-01
ld_date = Date("January 1, 1996")
```

Example 2 These statements assign the parent of the current control to a window variable:

```
window lw_current_window
lw_current_window = Parent
```

Example 3 This statement makes a CheckBox invisible:

```
cbk_on.Visible = FALSE
```

Example 4 This statement is not an assignment—it tests the value of the string in the SingleLineEdit sle_emp:

```
IF sle_emp.Text = "N" THEN Open(win_1)
```

Example 5 These statements concatenate two strings and assign the value to the string *Text1*:

```
string Text1
Text1 = sle_emp.Text+".DAT"
```

Example 6 These assignments use operator shortcuts:

```
int i = 4
i ++      // i is now 5.
i --      // i is 4 again.
i += 10   // i is now 14.
i /= 2    // i is now 7.
```

These shortcuts can be used only in pure assignment statements. They cannot be used with other operators in a statement. For example, the following is invalid:

```
int i, j
i = 12
j = i ++     // INVALID
```

The following is valid, because ++ is used by itself in the assignment:

```
int i, j
i = 12
i ++
j = i
```

CALL

Description

Calls an ancestor script from a script for a descendent object. You can call scripts for events in an ancestor of the user object, menu, or window. You can also call scripts for events for controls in an ancestor of the user object or window.

When you use the CALL statement to call an ancestor event script, the AncestorReturnValue variable is generated. For more information on the AncestorReturnValue variable, see “About events” on page 171.

Syntax

CALL *ancestorobject* {`*controlname*}::*event*

Parameter	**Description**
ancestorobject	An ancestor of the descendent object
controlname (optional)	The name of a control in an ancestor window or custom user object
event	An event in the ancestor object

Usage

Using the standard syntax
For most purposes, you should use the standard syntax for calling functions and events. For more information about the standard syntax, see "Syntax for calling PocketBuilder functions and events" on page 104.

The standard syntax allows you to trigger or post an event or function in an ancestor and then pass arguments, but it does not allow you to call a script for a control in the ancestor.

In some circumstances, you can use the pronoun Super when *ancestorobject* is the descendant object's immediate ancestor. See the discussion of "Super pronoun" on page 14.

If the call is being made to an ancestor event, the arguments passed to the current event are automatically propagated to the ancestor event. If you call a non-ancestor event and pass arguments, you need to use the new syntax, otherwise null will be passed for each argument.

Examples

Example 1 This statement calls a script for an event in an ancestor window:

```
CALL w_emp::Open
```

Example 2 This statement calls a script for an event in a control in an ancestor window:

```
CALL w_emp`cb_close::Clicked
```

CHOOSE CASE

Description

A control structure that directs program execution based on the value of a test expression (usually a variable).

Syntax

```
CHOOSE CASE testexpression
CASE expressionlist
   statementblock
{ CASE expressionlist
   statementblock
. . .
CASE expressionlist
   statementblock }
CASE ELSE
   statementblock }
END CHOOSE
```

Parameter	Description
testexpression	The expression on which you want to base the execution of the script
expressionlist	One of the following expressions: • A single value • A list of values separated by commas (such as 2, 4, 6, 8) • A TO clause (such as `1 TO 30`) • IS followed by a relational operator and comparison value (such as `IS>5`) • Any combination of the above with an implied OR between expressions (such as 1, 3, 5, 7, 9, 27 TO 33, `IS >42`)
statementblock	The block of statements you want PocketBuilder to execute if the test expression matches the value in *expressionlist*

Usage

At least one CASE clause is required. You must end a CHOOSE CASE control structure with END CHOOSE.

If *testexpression* at the beginning of the CHOOSE CASE statement matches a value in *expressionlist* for a CASE clause, the statements immediately following the CASE clause are executed. Control then passes to the first statement after the END CHOOSE clause.

If multiple CASE expressions exist, then *testexpression* is compared to each *expressionlist* until a match is found or the CASE ELSE or END CHOOSE is encountered.

If there is a CASE ELSE clause and the test value does not match any of the expressions, *statementblock* in the CASE ELSE clause is executed. If no CASE ELSE clause exists and a match is not found, the first statement after the END CHOOSE clause is executed.

Examples

Example 1 These statements provide different processing based on the value of the variable *Weight*:

```
CHOOSE CASE Weight
CASE IS<16
       Postage=Weight*0.30
       Method="USPS"
CASE 16 to 48
       Postage=4.50
       Method="UPS"
```

```
CASE ELSE
      Postage=25.00
      Method="FedEx"
END CHOOSE
```

Example 2 These statements convert the text in a SingleLineEdit control to a real value and provide different processing based on its value:

```
CHOOSE CASE Real(sle_real.Text)
CASE is < 10.99999
      sle_message.Text = "Real Case < 10.99999"
CASE 11.00 to 48.99999
      sle_message.Text = "Real Case 11 to 48.9999
CASE is > 48.9999
      sle_message.Text = "Real Case > 48.9999"
CASE ELSE
      sle_message.Text = "Cannot evaluate!"
END CHOOSE
```

CONTINUE

Description

In a DO...LOOP or a FOR...NEXT control structure, skips statements in the loop. CONTINUE takes no parameters.

Syntax

CONTINUE

Usage

When PocketBuilder encounters a CONTINUE statement in a DO...LOOP or FOR...NEXT block, control passes to the next LOOP or NEXT statement. The statements between the CONTINUE statement and the loop's end statement are skipped in the current iteration of the loop. In a nested loop, a CONTINUE statement bypasses statements in the *current* loop structure.

For information on how to break out of the loop, see EXIT on page 126.

Examples

Example 1 These statements display a message box twice: when *B* equals 2 and when *B* equals 3. As soon as *B* is greater than 3, the statement following CONTINUE is skipped during each iteration of the loop:

```
integer A=1, B=1
DO WHILE A < 100
      A = A+1
      B = B+1
      IF B > 3 THEN CONTINUE
      MessageBox("Hi", "B is " + String(B) )
LOOP
```

Example 2 These statements stop incrementing *B* as soon as *Count* is greater than 15:

```
integer A=0, B=0, Count
FOR Count = 1 to 100
      A = A + 1
      IF Count > 15 THEN CONTINUE
      B = B + 1
NEXT
// Upon completion, a=100 and b=15.
```

CREATE

Description

Creates an object instance for a specified object type. After a CREATE statement, properties of the created object instance can be referenced using dot notation.

The CREATE statement returns an object instance that can be stored in a variable of the same type.

Syntax 1 specifies the object type at compilation. Syntax 2 allows the application to choose the object type dynamically.

Syntax

Syntax 1 (specifies the object type at compilation):

objectvariable = CREATE *objecttype*

Parameter	**Description**
objectvariable	A global, instance, or local variable whose datatype is *objecttype*
objecttype	The object datatype

Syntax 2 (allows the application to choose the object type dynamically):

objectvariable = CREATE USING *objecttypestring*

Parameter	**Description**
objectvariable	A global, instance, or local variable whose datatype is the same class as the object being created or an ancestor of that class
objecttypestring	A string whose value is the name of the class datatype to be created

Usage

Use CREATE as the first reference to any class user object. This includes standard class user objects such as mailSession or Transaction.

The system provides one instance of several standard class user objects: Message, Error, Transaction, DynamicDescriptionArea, and DynamicStagingArea. You only need to use CREATE if you declare additional instances of these objects.

If you need a menu that is not part of an open window definition, use CREATE to create an instance of the menu. (See the function PopMenu on page 767.)

To create an instance of a visual user object or window, use the appropriate Open function (instead of CREATE).

You do not need to use CREATE to allocate memory for:

- A standard datatype, such as integer or string
- Any structure, such as the Environment object
- Any object whose AutoInstantiate setting is true
- Any object that has been instantiated using a function, such as Open

Specifying the object type dynamically CREATE USING allows your application to choose the object type dynamically. It is usually used to instantiate an ancestor variable with an instance of one of its descendants. The particular descendant is chosen at execution time.

For example, if uo_a has two descendants, uo_a_desc1 and uo_a_desc2, then the application can select the object to be created based on current conditions:

```
uo_a uo_a_var
string ls_objectname

IF ... THEN
      ls_objectname = "uo_a_desc1"
ELSE
      ls_objectname = "uo_a_desc2"
END IF
uo_a_var = CREATE USING ls_objectname
```

Destroying objects you create When you have finished with an object you created, you can call DESTROY to release its memory. However, you should call DESTROY only if you are sure that the object is not referenced by any other object. PocketBuilder's garbage collection mechanism maintains a count of references to each object and destroys unreferenced objects automatically.

For more information about garbage collection, see "Garbage collection" on page 78.

Examples

Example 1 These statements create a new transaction object and stores the object in the variable DBTrans:

```
transaction DBTrans
DBTrans = CREATE transaction
DBTrans.DBMS = 'ODBC'
```

Example 2 These statements create a user object when the application has need of the services it provides. Because the user object might or might not exist, the code that accesses it checks whether it exists before calling its functions.

The object that creates the service object declares *invo_service* as an instance variable:

```
n_service invo_service
```

The Open event for the object creates the service object:

```
//Open event of some object
IF (some condition) THEN
   invo_service = CREATE n_service
END IF
```

When another script wants to call a function that belongs to the n_service class, it verifies that *invo_service* is instantiated:

```
IF IsValid(invo_service) THEN
   invo_service.of_perform_some_work( )
END IF
```

If the service object was created, then it also needs to be destroyed:

```
IF isvalid(invo_service) THEN DESTROY invo_service
```

Example 3 When you create a DataStore object, you also have to give it a DataObject and call SetTransObject before you can use it:

```
l_ds_delete = CREATE u_ds
l_ds_delete.DataObject = 'd_user_delete'
l_ds_delete.SetTransObject(SQLCA)
li_cnt = l_ds_delete.Retrieve(lstr_data.name)
```

Example 4 In this example, n_file_service_class is an ancestor object, and n_file_service_class_ansi and n_file_service_class_dbcs are its descendants. They hold functions and variables that provide services for the application. The code chooses which object to create based on whether the user is running in a DBCS environment:

```
n_file_service_class  lnv_fileservice
string ls_objectname
```

```
environment luo_env

GetEnvironment ( luo_env )
IF luo_env.charset = charsetdbcs! THEN
   ls_objectname = "n_file_service_class_dbcs"
ELSE
   ls_objectname = "n_file_service_class_ansi"
END IF

lnv_fileservice = CREATE USING ls_objectname
```

DESTROY

Description

Eliminates an object instance that was created with the CREATE statement. After a DESTROY statement, properties of the deleted object instance can no longer be referenced.

Syntax

DESTROY *objectvariable*

Parameter	Description
objectvariable	A variable whose datatype is a PocketBuilder object

Usage

When you are finished with an object that you created, you can call DESTROY to release its memory. However, you should call DESTROY only if you are sure that the object is not referenced by any other object. PocketBuilder's garbage collection mechanism maintains a count of references to each object and destroys unreferenced objects automatically.

For more information about garbage collection, see "Garbage collection" on page 78.

All objects are destroyed automatically when your application terminates.

Examples

Example 1 The following statement destroys the transaction object DBTrans that was created with a CREATE statement:

```
DESTROY DBTrans
```

Example 2 This example creates an OLEStorage variable *istg_prod_pic* in a window's Open event. When the window is closed, the Close event script destroys the object. The variable's declaration is:

```
OLEStorage istg_prod_pic
```

The window’s Open event creates an object instance and opens an OLE storage file:

```
integer li_result
istg_prod_pic = CREATE OLEStorage
li_result = stg_prod_pic.Open("PICTURES.OLE")
```

The window’s Close event destroys *istg_prod_pic*:

```
integer li_result
li_result = istg_prod_pic.Save( )
IF li_result = 0 THEN
      DESTROY istg_prod_pic
END IF
```

DO...LOOP

Description

A control structure that is a general-purpose iteration statement used to execute a block of statements while or until a condition is true.

DO... LOOP has four formats:

- **DO UNTIL** Executes a block of statements until the specified condition is true. If the condition is true on the first evaluation, the statement block does not execute.
- **DO WHILE** Executes a block of statements while the specified condition is true. The loop ends when the condition becomes false. If the condition is false on the first evaluation, the statement block does not execute.
- **LOOP UNTIL** Executes a block of statements at least once and continues until the specified condition is true.
- **LOOP WHILE** Executes a block of statements at least once and continues while the specified condition is true. The loop ends when the condition becomes false.

In all four formats of the DO...LOOP control structure, DO marks the beginning of the statement block that you want to repeat. The LOOP statement marks the end.

You can nest DO...LOOP control structures.

Syntax

DO UNTIL *condition*
  statementblock
LOOP

```
DO WHILE condition
  statementblock
LOOP
```

```
DO
  statementblock
LOOP UNTIL condition
```

```
DO
  statementblock
LOOP WHILE condition
```

Parameter	Description
condition	The condition you are testing
statementblock	The block of statements you want to repeat

Usage

Use DO WHILE or DO UNTIL when you want to execute a block of statements *only* if a condition is true (for WHILE) or false (for UNTIL). DO WHILE and DO UNTIL test the condition *before* executing the block of statements.

Use LOOP WHILE or LOOP UNTIL when you want to execute a block of statements *at least once*. LOOP WHILE and LOOP UNTIL test the condition *after* the block of statements has been executed.

Examples

DO UNTIL The following DO UNTIL repeatedly executes the Beep function until *A* is greater than 15:

```
integer A = 1, B = 1

DO UNTIL A > 15
       Beep(A)
       A = (A + 1) * B

LOOP
```

DO WHILE The following DO WHILE repeatedly executes the Beep function only while *A* is less than or equal to 15:

```
integer A = 1, B = 1

DO WHILE A <= 15
       Beep(A)
       A = (A + 1) * B

LOOP
```

LOOP UNTIL The following LOOP UNTIL executes the Beep function and then continues to execute the function until *A* is greater than 1:

```
integer A = 1, B = 1
DO
       Beep(A)
```

```
      A = (A + 1) * B
LOOP UNTIL A > 15
```

LOOP WHILE The following LOOP WHILE repeatedly executes the Beep function while *A* is less than or equal to 15:

```
integer A = 1, B = 1

DO
      Beep(A)
      A = (A + 1) * B

LOOP WHILE A <= 15
```

EXIT

Description

In a DO...LOOP or a FOR...NEXT control structure, passes control out of the current loop. EXIT takes no parameters.

Syntax

EXIT

Usage

An EXIT statement in a DO...LOOP or FOR...NEXT control structure causes control to pass to the statement following the LOOP or NEXT statement. In a nested loop, an EXIT statement passes control out of the *current* loop structure.

For information on how to jump to the end of the loop and continue looping, see CONTINUE on page 119.

Examples

Example 1 This EXIT statement causes the loop to terminate if an element in the *Nbr* array equals 0:

```
int Nbr[10]
int Count = 1
// Assume values get assigned to Nbr array...

DO WHILE Count < 11
      IF Nbr[Count] = 0 THEN EXIT
      Count = Count + 1
LOOP

MessageBox("Hi",  "Count is now "  + String(Count) )
```

Example 2 This EXIT statement causes the loop to terminate if an element in the *Nbr* array equals 0:

```
int Nbr[10]
int Count
// Assume values get assigned to Nbr array...
```

```
FOR Count = 1 to 10
      IF Nbr[Count] = 0 THEN EXIT
NEXT

MessageBox("Hi",  "Count is now "  + String(Count) )
```

FOR...NEXT

Description

A control structure that is a numerical iteration, used to execute one or more statements a specified number of times.

Syntax

FOR *varname* = *start* TO *end* {STEP *increment*}
  statementblock
NEXT

Parameter	Description
varname	The name of the iteration counter variable. It can be any numerical type (integer, double, real, long, or decimal), but integers provide the fastest performance.
start	Starting value of *varname*.
end	Ending value of *varname*.
increment (optional)	The increment value. *Increment* must be a constant and the same datatype as *varname*. If you enter an increment, STEP is required. +1 is the default increment.
statementblock	The block of statements you want to repeat.

Usage

Using the *start* and *end* parameters For a positive *increment*, *end* must be greater than *start*. For a negative *increment*, *end* must be less than *start*.

When *increment* is positive and *start* is greater than *end*, *statementblock* does not execute. When *increment* is negative and *start* is less than *end*, *statementblock* does not execute.

When *start* and *end* are expressions, they are reevaluated on each pass through the loop. If the expression's value changes, it affects the number of loops. Consider this example—the body of the loop changes the number of rows, which changes the result of the RowCount function:

```
FOR n = 1 TO dw_1.RowCount( )
      dw_1.DeleteRow(1)
NEXT
```

A variable as the step increment
If you need to use a variable for the step increment, you can use one of the DO...LOOP constructions and increment the counter yourself within the loop.

Nesting You can nest FOR...NEXT statements. You must have a NEXT for each FOR.

You can end the FOR loop with the keywords END FOR instead of NEXT.

Avoid overflow
If *start* or *end* is too large for the datatype of *varname*, *varname* will overflow, which might create an infinite loop. Consider this statement for the integer *li_int*:

```
FOR li_int = 1 TO 50000
```

The end value 50000 is too large for an integer. When *li_int* is incremented, it overflows to a negative value before reaching 50000, creating an infinite loop.

Examples

Example 1 These statements add 10 to *A* as long as *n* is >=5 and <=25:

```
FOR n = 5 to 25
      A = A+10
NEXT
```

Example 2 These statements add 10 to *A* and increment n by 5 as long as *n* is >= 5 and <=25:

```
FOR N = 5 TO 25 STEP 5
      A = A+10
NEXT
```

Example 3 These statements contain two lines that will never execute because *increment* is negative and *start* is less than *end*:

```
FOR Count = 1 TO 100 STEP -1
   IF Count < 1 THEN EXIT // These 2 lines
   Box[Count] = 10        // will never execute.
NEXT
```

Example 4 These are nested FOR...NEXT statements:

```
Int Matrix[100,50,200]
FOR i = 1 to 100
      FOR j = 1 to 50
      FOR k = 1 to 200
         Matrix[i,j,k]=1
      NEXT
      NEXT
NEXT
```

GOTO

Description

Transfers control from one statement in a script to another statement that is labeled.

Syntax

GOTO *label*

Parameter	Description
label	The label associated with the statement to which you want to transfer control. A label is an identifier followed by a colon (such as OK:). Do not use the colon with a label in the GOTO statement.

Examples

Example 1 This GOTO statement skips over the `Taxable=FALSE` line:

```
Goto NextStep
Taxable=FALSE    //This statement never executes.
NextStep:
Rate=Count/Count4
```

Example 2 This GOTO statement transfers control to the statement associated with the label OK:

```
GOTO OK
.
.
.
OK:
.
.
.
```

HALT

Description

Terminates an application.

Syntax

HALT {CLOSE}

Usage

When PocketBuilder encounters Halt without the keyword CLOSE, it immediately terminates the application.

When PocketBuilder encounters Halt with the keyword CLOSE, it immediately executes the script for the Close event for the application and then terminates the application. If there is no script for the Close event at the application level, PocketBuilder immediately terminates the application.

Examples

Example 1 This statement stops the application if the user enters a password in the SingleLineEdit named sle_password that does not match the value stored in a string named *CorrectPassword*:

```
IF sle_password.Text <> CorrectPassword THEN HALT
```

Example 2 This statement executes the script for the Close event for the application before it terminates the application if the user enters a password in sle_password that does not match the value stored in the string *CorrectPassword*:

```
IF sle_password.Text <> CorrectPassword  &
   THEN HALT CLOSE
```

IF...THEN

Description

A control structure used to cause a script to perform a specified action if a stated condition is true. Syntax 1 uses a single-line format, and Syntax 2 uses a multiline format.

Syntax

Syntax 1 (the single-line format):

IF *condition* THEN *action1* {ELSE *action2*}

Parameter	Description
condition	The condition you want to test.

Parameter	Description
action1	The action you want performed if the condition is true. The action must be a single statement on the same line as the rest of the IF statement.
action2 (optional)	The action you want performed if the condition is false. The action must be a single statement on the same line as the rest of the IF statement.

Syntax 2 (the multiline format):

```
IF condition1 THEN
    action1
{ ELSEIF condition2 THEN
    action2
. . . }
{ ELSE
    action3 }
END IF
```

Parameter	Description
condition1	The first condition you want to test.
action1	The action you want performed if *condition1* is true. The action can be a statement or multiple statements that are separated by semicolons or placed on separate lines. At least one action is required.
condition2 (optional)	The condition you want to test if *condition1* is false. You can have multiple ELSEIF...THEN statements in an IF...THEN control structure.
action2	The action you want performed if *condition2* is true. The action can be a statement or multiple statements that are separated by semicolons or placed on separate lines.
action3 (optional)	The action you want performed if none of the preceding conditions is true. The action can be a statement or multiple statements that are separated by semicolons or placed on separate lines.

Usage

You can use continuation characters to place the single-line format on more than one physical line in the script.

You must end a multiline IF...THEN control structure with END IF (which is two words).

Examples

Example 1 This single-line IF...THEN statement opens window w_first if *Num* is equal to 1; otherwise, w_rest is opened:

```
IF Num = 1 THEN Open(w_first) ELSE Open(w_rest)
```

Example 2 This single-line IF...THEN statement displays a message if the value in the SingleLineEdit sle_State is “TX”. It uses the continuation character to continue the single-line statement across two physical lines in the script:

```
IF sle_State.text="TX" THEN   &
   MessageBox("Hello","Tex")
```

Example 3 This multiline IF...THEN compares the horizontal positions of windows w_first and w_second. If w_first is to the right of w_second, w_first is moved to the left side of the screen:

```
IF w_first.X > w_second.X THEN
   w_first.X = 0
END IF
```

Example 4 This multiline IF...THEN causes the application to:

- Beep twice if X equals Y
- Display the Parts list box and highlight item 5 if X equals Z
- Display the Choose list box if X is blank
- Hide the Empty button and display the Full button if none of the above conditions is true

```
IF X=Y THEN
   Beep(2)
ELSEIF X=Z THEN
   Show (lb_parts); lb_parts.SetState(5,TRUE)
ELSEIF X=" " THEN
   Show (lb_choose)
ELSE
   Hide(cb_empty)
   Show(cb_full)
END IF
```

RETURN

Description Stops the execution of a script or function immediately.

Syntax RETURN { *expression* }

Parameter	Description
expression	In a function, any value (or expression) you want the function to return. The return value must be the datatype specified as the return type in the function.

Usage

When a user's action triggers an event and PocketBuilder encounters RETURN in the event script, it terminates execution of that script immediately and waits for the next user action.

When a script calls a function or event and PocketBuilder encounters RETURN in the code, RETURN transfers (returns) control to the point at which the function or event was called.

Examples

Example 1 This script causes the system to beep once; the second beep statement will not execute:

```
Beep(1)
RETURN
Beep(1)  // This statement will not execute.
```

Example 2 These statements in a user-defined function return the result of dividing *Arg1* by *Arg2* if *Arg2* is not equal to zero; they return -1 if *Arg2* is equal to zero:

```
IF Arg2 <> 0 THEN
   RETURN Arg1/Arg2
ELSE
   RETURN -1
END IF
```

THROW

Description

Used to manually trigger exception handling for user-defined exceptions.

Syntax

THROW *exlvalue*

Parameter	Description
exlvalue	Variable (or expression that evaluates to a valid instance of an object) of type Throwable. Usually the object type thrown is a user-defined exception class derived from the system Exception class that inherits from Throwable.

Usage

The variable following the THROW reserved word must be a valid object instance or an expression that produces a valid object instance that derives from the Throwable datatype. For example, you can use an expression such as:

```
THROW create ExceptionType
```

where *ExceptionType* is an object of type Throwable.

If you attempt to throw a noninstantiated exception, you will not get back the exception information you want, since the only exception information you retrieve will be a NullObjectError.

In a method script, you can only throw an exception that you declare in the method prototype or that you handle in a try-catch block. The PowerScript compiler displays an error message if you try to throw a user-defined exception without declaring it in the prototype Throws statement and without surrounding it in an appropriate try-catch block.

When a RuntimeError, or a descendant of RuntimeError, is thrown, the instance variable containing line number information will be filled in at the point where the THROW statement occurs. If the error is handled and thrown again, this information will not be updated unless it has specifically been set to null.

Examples

```
long ll_result
ll_result = myConnection.ConnectToServer()

   ConnectionException ex
   ex = create ConnectionException
   ex.connectResult = ll_result
   THROW ex
end if
```

THROWS

Description

Used to declare the type of exception that a method triggers. It is part of the method prototype.

Syntax

methodname ({*arguments*}) THROWS *ExceptionType* { , *ExceptionType*, ... }

Parameter	Description
methodname	Name of the method that throws an exception.
arguments	Arguments of the method that throws an exception. Depending on the method, the method arguments can be optional.

Parameter	Description
ExceptionType	Object of type Throwable. Usually the object type thrown is a user-defined exception class derived from the system Exception class. If you define multiple potential exceptions for a method, you can throw each type of exception in the same clause by separating the exception types with commas.

Usage

Internal use only.

You do not type or otherwise add the THROWS clause to function calls in a PocketBuilder script. However, you can add a THROWS clause to any PocketBuilder function or to any user event that is not defined by a pbm event ID.

For more information about adding a THROWS clause to a function or event prototype, see the *Users Guide*. For more information about exception handling, see the *Resource Guide*.

TRY...CATCH...FINALLY...END TRY

Description

Isolates code that can cause an exception, describes what to do if an exception of a given type is encountered, and allows you to close files or network connections (and return objects to their original state) whether or not an exception is encountered.

Syntax

```
TRY
  trystatements
CATCH ( ThrowableType1 exIdentifier1 )
  catchstatements1
CATCH ( ThrowableType2 exIdentifier2 )
  catchstatements2
...
CATCH ( ThrowableTypeN exIdentifierN )
  catchstatementsN
FINALLY
  cleanupstatements
END TRY
```

Parameter	Description
trystatements	Block of code that might potentially throw an exception.

Parameter	**Description**
ThrowableTypeN	Object type of exception to be caught. A CATCH block is optional if you include a FINALLY block. You can include multiple CATCH blocks. Every CATCH block in a try-catch block must include a corresponding exception object type and a local variable of that type.
exIdentifierN	Local variable of type *ThrowableTypeN*.
catchstatementsN	Code to handle the exception being caught.
cleanupstatements	Cleanup code. The FINALLY block is optional if you include one or more CATCH block.

Usage

The TRY block, which is the block of statements between the TRY and CATCH keywords (or the TRY and FINALLY keywords if there is no CATCH clause), is used to isolate code that might potentially throw an exception. The statements in the TRY block are run unconditionally until either the entire block of statements is executed or some statement in the block causes an exception to be thrown.

Use a CATCH block or multiple CATCH blocks to handle exceptions thrown in a TRY block. In the event that an exception is thrown, execution of the TRY block is stopped and the statements in the first CATCH block are executed—if and only if the exception thrown is of the same type or a descendant of the type of the identifier following the CATCH keyword.

If the exception thrown is not the same type or a descendant type of the identifier in the first CATCH block, the exception is not handled by this CATCH block. If there are additional CATCH blocks, they are evaluated in the order they appear. If the exception cannot be handled by any of the CATCH blocks, the statements in the FINALLY block are executed.

The exception then continues to unwind the call stack to any outer nested try-catch blocks. If there are no outer nested blocks, the SystemError event on the Application object is fired.

If no exception is thrown, execution continues at the beginning of the FINALLY block if one exists; otherwise, execution continues on the line following the END TRY statement.

See also

THROW

CHAPTER 8

SQL Statements

About this chapter

This chapter describes the embedded SQL and dynamic SQL statements and how to use them in scripts.

Contents

Using SQL in scripts

PowerScript supports standard embedded SQL statements and dynamic SQL statements in scripts. In general, PowerScript supports all DBMS-specific clauses and reserved words that occur in the supported SQL statements. For example, PocketBuilder supports DBMS-specific built-in functions within a SELECT command.

For information about embedded SQL, see online Help.

Referencing PowerScript variables in scripts

Wherever constants can be referenced in SQL statements, PowerScript variables preceded by a colon (:) can be substituted. Any valid PowerScript variable can be used. This INSERT statement uses a constant value:

```
INSERT INTO EMPLOYEE ( SALARY )
       VALUES ( 18900 ) ;
```

The same statement using a PowerScript variable to reference the constant might look like this:

```
int   Sal_var
Sal_var = 18900
INSERT INTO EMPLOYEE ( SALARY )
       VALUES ( :Sal_var ) ;
```

Using indicator variables

PocketBuilder supports indicator variables, which are used to identify null values or conversion errors after a database retrieval. Indicator variables are integers that are specified in the *HostVariableList* of a FETCH or SELECT statement.

Each indicator variable is separated from the variable it is indicating by a space (but no comma). For example, this statement is a *HostVariableList* without indicator variables:

```
:Name, :Address, :City
```

The same *HostVariableList* with indicator variables looks like this:

```
:Name :IndVar1, :Address :IndVar2, :City :IndVar3
```

Indicator variables have one of these values:

Page	Meaning
0	Valid, non-null value
-1	Null value
-2	Conversion error

Error reporting
Not all DBMSs return a conversion error when the datatype of a column does not match the datatype of the associated variable.

The following statement uses the indicator variable *IndVar2* to see if Address contains a null value:

```
if IndVar2 = -1 then...
```

You can also use the PowerScript IsNull function to accomplish the same result without using indicator variables:

```
if IsNull( Address ) then ...
```

This statement uses the indicator variable *IndVar3* to set City to null:

```
IndVar3 = -1
```

You can also use the PowerScript SetNull function to accomplish the same result without using indicator variables:

```
SetNull( City )
```

Error handling in scripts

The scripts shown in the SQL examples above do not include error handling, but it is good practice to test the success and failure codes (the SQLCode attribute) in the transaction object after *every* statement. The codes are:

Value	Meaning
0	Success.
100	Fetched row not found.
-1	Error; the statement failed. Use SQLErrText or SQLDBCode to obtain the detail.

After certain statements, such as DELETE, FETCH, and UPDATE, you should also check the SQLNRows property of the transaction object to make sure the action affected at least one row.

About SQLErrText and SQLDBCode The string SQLErrText in the transaction object contains the database vendor-supplied error message. The long named SQLDBCode in the transaction object contains the database vendor-supplied status code:

```
IF SQLCA.SQLCode = -1 THEN
      MessageBox("SQL error", SQLCA.SQLErrText)
END IF
```

Painting standard SQL

You can paint the following SQL statements in scripts and functions:

- Declarations of SQL cursors and stored procedures
- Cursor FETCH, UPDATE, and DELETE statements
- Noncursor SELECT, INSERT, UPDATE, and DELETE statements

For more information about scope, see “Where to declare variables” on page 31.

You can declare cursors and stored procedures at the scope of global, instance, shared, or local variables. A cursor or procedure can be declared in the Script view using the Paste SQL button in the PainterBar.

You can paint standard embedded SQL statements in the Script view, the Function painter, and the Interactive SQL view in the Database painter using the Paste SQL button in the PainterBar or the Paste Special>SQL item from the pop-up menu.

Supported SQL statements

In general, all DBMS-specific features are supported in PowerScript if they occur within a PowerScript-supported SQL statement. For example, PowerScript supports DBMS-specific built-in functions within a SELECT command.

However, any SQL statement that contains a SELECT clause must also contain a FROM clause in order for the script to compile successfully. To solve this problem, add a FROM clause that uses a "dummy" table to SELECT statements without FROM clauses. For example:

```
string res
select user_name() into:res from dummy;
select db_name() into:res from dummy;
select date('2001-01-02:21:20:53') into:res from dummy;
```

CLOSE Cursor

Description

Closes the SQL cursor *CursorName*; ends processing of *CursorName*.

Syntax

CLOSE *CursorName* ;

Parameter	Description
CursorName	The cursor you want to close

Usage

This statement must be preceded by an OPEN statement for the same cursor. The USING TransactionObject clause is not allowed with CLOSE; the transaction object was specified in the statement that declared the cursor.

CLOSE often appears in the script that is executed when the SQL code after a fetch equals 100 (not found).

Error handling
It is good practice to test the success/failure code after executing a CLOSE cursor statement.

Examples

This statement closes the *Emp_cursor* cursor:

```
CLOSE Emp_cursor ;
```

CLOSE Procedure

Description

Closes the SQL procedure *ProcedureName*; ends processing of *ProcedureName*.

DBMS-specific
Not all DBMSs support stored procedures.

Syntax

CLOSE *ProcedureName*;

Parameter	Description
ProcedureName	The stored procedure you want to close

Usage

This statement must be preceded by an EXECUTE statement for the same procedure. The USING TransactionObject clause is not allowed with CLOSE; the transaction object was specified in the statement that declared the procedure.

Use CLOSE only to close procedures that return result sets. PocketBuilder automatically closes procedures that do not return result sets (and sets the return code to 100).

CLOSE often appears in the script that is executed when the SQL code after a fetch equals 100 (not found).

Error handling
It is good practice to test the success/failure code after executing a CLOSE Procedure statement.

Examples

This statement closes the stored procedure named *Emp_proc*:

```
CLOSE Emp_proc ;
```

COMMIT

Description

Permanently updates all database operations since the previous COMMIT, ROLLBACK, or CONNECT for the specified transaction object.

Using COMMIT and ROLLBACK in a server component
Server component connections are not supported in PocketBuilder. For information on COMMIT and ROLLBACK commands embedded in a server component, see *Connecting to Your Database* and *Application Techniques* in the PowerBuilder documentation set.

Syntax

COMMIT {USING *TransactionObject*};

Parameter	Description
TransactionObject	The name of the transaction object for which you want to permanently update all database operations since the previous COMMIT, ROLLBACK, or CONNECT. This clause is required only for transaction objects other than the default (SQLCA).

Usage

COMMIT does not cause a disconnect, but it does close all open cursors or procedures. (But note that the DISCONNECT statement in PocketBuilder does issue a COMMIT.)

Error handling
It is good practice to test the success/failure code after executing a COMMIT statement.

Examples

Example 1 This statement commits all operations for the database specified in the default transaction object:

```
COMMIT ;
```

Example 2 This statement commits all operations for the database specified in the transaction object named *Emp_tran*:

```
COMMIT USING Emp_tran ;
```

CONNECT

Description

Connects to a specified database.

Syntax

CONNECT {USING *TransactionObject*};

Parameter	Description
TransactionObject	The name of the transaction object containing the required connection information for the database to which you want to connect. This clause is required only for transaction objects other than the default (SQLCA).

Usage

This statement must be executed before any actions (such as INSERT, UPDATE, or DELETE) can be processed using the default transaction object or the specified transaction object.

Error handling
It is good practice to test the success/failure code after executing a CONNECT statement.

Examples

Example 1 This statement connects to the database specified in the default transaction object:

```
CONNECT ;
```

Example 2 This statement connects to the database specified in the transaction object named *Emp_tran*:

```
CONNECT USING Emp_tran ;
```

DECLARE Cursor

Description

Declares a cursor for the specified transaction object.

Syntax

DECLARE *CursorName* CURSOR FOR *SelectStatement*
{USING *TransactionObject*};

Parameter	Description
CursorName	Any valid PocketBuilder name.
SelectStatement	Any valid SELECT statement.
TransactionObject	The name of the transaction object for which you want to declare the cursor. This clause is required only for transaction objects other than the default (SQLCA).

Usage

DECLARE Cursor is a nonexecutable command and is analogous to declaring a variable.

To declare a local cursor, open the script in the Script view and select Paste SQL from the PainterBar or the Edit>Paste Special menu. To declare a global, instance, or shared cursor, select Declare from the first drop-down list in the Script view and Global Variables, Instance Variables, or Shared Variables from the second drop-down list, then select Paste SQL.

For information about global, instance, shared, and local scope, see “Where to declare variables” on page 31.

Examples

This statement declares the cursor called *Emp_cur* for the database specified in the default transaction object. It also references the *Sal_var* variable, which must be set to an appropriate value before you execute the OPEN *Emp_cur* command:

```
DECLARE Emp_cur CURSOR FOR
      SELECT employee.emp_number, employee.emp_name
      FROM employee
      WHERE employee.emp_salary > :Sal_var ;
```

DECLARE Procedure

Description

Declares a procedure for the specified transaction object.

DBMS-specific
Not all DBMSs support stored procedures.

Syntax

DECLARE *ProcedureName* PROCEDURE FOR
    StoredProcedureName
    @Param1=Value1, *@Param2=Value2*,...
    {USING *TransactionObject*};

Parameter	Description
ProcedureName	Any valid PocketBuilder name.
StoredProcedureName	Any stored procedure in the database.
@Paramn=Valuen	The name of a parameter (argument) defined in the stored procedure and a valid PocketBuilder expression; represents the number of the parameter and value.
TransactionObject	The name of the transaction object for which you want to declare the procedure. This clause is required only for transaction objects other than the default (SQLCA).

Usage

DECLARE Procedure is a nonexecutable command. It is analogous to declaring a variable.

To declare a local procedure, open the script in the Script view and select Paste SQL from the PainterBar or the Edit>Paste Special menu. To declare a global, instance, or shared procedure, select Declare from the first drop-down list in the Script view and Global Variables, Instance Variables, or Shared Variables from the second drop-down list, then select Paste SQL.

For information about global, instance, shared, and local scope, see "Where to declare variables" on page 31.

Examples

Example 1 This statement declares the Sybase ASE procedure *Emp_proc* for the database specified in the default transaction object. It references the *Emp_name_var* and *Emp_sal_var* variables, which must be set to appropriate values before you execute the EXECUTE Emp_proc command:

```
DECLARE Emp_proc procedure for GetName
   @emp_name = :Emp_name_var,
   @emp_salary = :Emp_sal_var ;
```

Example 2 This statement declares the ORACLE procedure Emp_proc for the database specified in the default transaction object. It references the *Emp_name_var* and *Emp_sal_var* variables, which must be set to appropriate values before you execute the EXECUTE Emp_proc command:

```
DECLARE Emp_proc procedure for GetName
(:Emp_name_var, :Emp_sal_var) ;
```

DELETE

Description

Deletes the rows in *TableName* specified by *Criteria*.

Syntax

DELETE FROM *TableName* WHERE *Criteria* {USING *TransactionObject*};

Parameter	**Description**
TableName	The name of the table from which you want to delete rows.
Criteria	Criteria that specify which rows to delete.
TransactionObject	The name of the transaction object that identifies the database containing the table. This clause is required only for transaction objects other than the default (SQLCA).

Usage

Error handling
It is good practice to test the success/failure code after executing a DELETE statement. To see if the DELETE was successful, you can test SLQCode for a failure code. However, if nothing matches the WHERE clause and no rows are deleted, SQLCode is still set to zero. To make sure the delete affected at least one row, check the SQLNRows property of the transaction object.

Examples

Example 1 This statement deletes rows from the Employee table in the database specified in the default transaction object where Emp_num is less than 100:

```
DELETE FROM Employee WHERE Emp_num < 100 ;
```

Example 2 These statements delete rows from the Employee table in the database named in the transaction object named *Emp_tran* where *Emp_num* is equal to the value entered in the SingleLineEdit sle_number:

```
int    Emp_num
Emp_num = Integer(sle_number.Text)
DELETE FROM Employee
       WHERE Employee.Emp_num = :Emp_num ;
```

The integer *Emp_num* requires a colon in front of it to indicate it is a variable when it is used in a WHERE clause.

DELETE Where Current of Cursor

Description

Deletes the row in which the cursor is positioned.

DBMS-specific
Not all DBMSs support DELETE Where Current of Cursor.

Syntax

DELETE FROM *TableName* WHERE CURRENT OF *CursorName*;

Parameter	Description
TableName	The name of the table from which you want to delete a row
CursorName	The name of the cursor in which the table was specified

Usage

The USING TransactionObject clause is not allowed with this form of DELETE Where Current of Cursor; the transaction object was specified in the statement that declared the cursor.

Error handling
It is good practice to test the success/failure code after executing a DELETE Where Current of Cursor statement.

Examples

This statement deletes from the Employee table the row in which the cursor named *Emp_cur* is positioned:

```
DELETE FROM Employee WHERE current of Emp_curs ;
```

DISCONNECT

Description

Executes a COMMIT for the specified transaction object and then disconnects from the specified database.

Syntax

DISCONNECT {USING *TransactionObject*};

Parameter	Description
TransactionObject	The name of the transaction object that identifies the database you want to disconnect from and in which you want to permanently update all database operations since the previous COMMIT, ROLLBACK, or CONNECT. This clause is required only for transaction objects other than the default (SQLCA).

Usage

Error handling
It is good practice to test the success/failure code after executing a DISCONNECT statement.

Examples

Example 1 This statement disconnects from the database specified in the default transaction object:

```
DISCONNECT ;
```

Example 2 This statement disconnects from the database specified in the transaction object named *Emp_tran*:

```
DISCONNECT USING Emp_tran ;
```

EXECUTE

Description

Executes the previously declared procedure identified by *ProcedureName*.

Syntax

EXECUTE *ProcedureName*;

Parameter	Description
ProcedureName	The name assigned in the DECLARE statement of the stored procedure you want to execute. The procedure must have been declared previously. *ProcedureName* is not necessarily the name of the procedure stored in the database.

Usage

The USING TransactionObject clause is not allowed with EXECUTE; the transaction object was specified in the statement that declared the procedure.

Error handling
It is good practice to test the success/failure code after executing an EXECUTE statement.

Examples

This statement executes the stored procedure *Emp_proc*:

```
EXECUTE Emp_proc ;
```

FETCH

Description

Fetches the row after the row on which *Cursor* | *Procedure* is positioned.

Syntax

FETCH *Cursor* | *Procedure* INTO *HostVariableList*;

Parameter	Description
Cursor or Procedure	The name of the cursor or procedure from which you want to fetch a row
HostVariableList	PowerScript variables into which data values will be retrieved

Usage

The USING TransactionObject clause is not allowed with FETCH; the transaction object was specified in the statement that declared the cursor or procedure.

If your DBMS supports formats of FETCH other than the customary (and default) FETCH NEXT, you can specify FETCH FIRST, FETCH PRIOR, or FETCH LAST.

Error handling
It is good practice to test the success/failure code after executing a FETCH statement. To see if the FETCH was successful, you can test SLQCode for a failure code. However, if nothing matches the WHERE clause and no rows are fetched, SQLCode is still set to 100. To make sure the fetch affected at least one row, check the SQLNRows property of the transaction object.

Examples

Example 1 This statement fetches data retrieved by the SELECT clause in the declaration of the cursor named *Emp_cur* and puts it into *Emp_num* and *Emp_name*:

```
int        Emp_num
string     Emp_name
FETCH Emp_cur INTO :Emp_num, :Emp_name ;
```

Example 2 If sle_emp_num and sle_emp_name are SingleLineEdits, these statements fetch from the cursor named *Emp_cur*, store the data in *Emp_num* and *Emp_name*, and then convert *Emp_num* from an integer to a string, and put them in *sle_emp_num* and *sle_emp_name*:

```
int        Emp_num
string     Emp_name
FETCH Emp_cur INTO :emp_num, :emp_name ;
sle_emp_num.Text = string(Emp_num)
sle_emp_name.Text = Emp_name
```

INSERT

Description

Inserts one or more new rows into the table specified in *RestOfInsertStatement*.

Syntax

INSERT *RestOfInsertStatement*
{USING *TransactionObject*} ;

Parameter	Description
RestOfInsertStatement	The rest of the INSERT statement (the INTO clause, list of columns and values or source).
TransactionObject	The name of the transaction object that identifies the database containing the table. This clause is required only for transaction objects other than the default (SQLCA).

Usage

Error handling
It is good practice to test the success/failure code after executing an INSERT statement.

Examples

Example 1 These statements insert a row with the values in *EmpNbr* and *EmpName* into the Emp_nbr and Emp_name columns of the Employee table identified in the default transaction object:

```
int EmpNbr
string EmpName
...
INSERT INTO Employee (employee.Emp_nbr,
      employee.Emp_name)
      VALUES (:EmpNbr, :EmpName) ;
```

Example 2 These statements insert a row with the values entered in the SingleLineEdits sle_number and sle_name into the Emp_nbr and Emp_name columns of the Employee table in the transaction object named *Emp_tran*:

```
int     EmpNbr
string   EmpName
EmpNbr = Integer(sle_number.Text)
EmpName = sle_name.Text
INSERT INTO Employee (employee.Emp_nbr,
      employee.Emp_name)
      VALUES (:EmpNbr, :EmpName) USING Emp_tran ;
```

OPEN Cursor

Description

Causes the SELECT specified when the cursor was declared to be executed.

Syntax

OPEN *CursorName* ;

Parameter	Description
CursorName	The name of the cursor you want to open

Usage

The USING TransactionObject clause is not allowed with OPEN; the transaction object was specified in the statement that declared the cursor.

Error handling
It is good practice to test the success/failure code after executing an OPEN Cursor statement.

Examples

This statement opens the cursor *Emp_curs*:

```
OPEN Emp_curs ;
```

ROLLBACK

Description

Cancels all database operations in the specified database since the last COMMIT, ROLLBACK, or CONNECT.

Using *COMMIT* and *ROLLBACK* in a server component
Server component connections are not supported in PocketBuilder. For information on COMMIT and ROLLBACK commands embedded in a server component, see *Connecting to Your Database* and *Application Techniques* in the PowerBuilder documentation set.

Syntax

ROLLBACK {USING *TransactionObject*} ;

Parameter	Description
TransactionObject	The name of the transaction object that identifies the database in which you want to cancel all operations since the last COMMIT, ROLLBACK, or CONNECT. This clause is required only for transaction objects other than the default (SQLCA).

Usage

ROLLBACK does not cause a disconnect, but it does close all open cursors and procedures.

Error handling
It is good practice to test the success/failure code after executing a ROLLBACK statement.

Examples

Example 1 This statement cancels all database operations in the database specified in the default transaction object:

```
ROLLBACK ;
```

Example 2 This statement cancels all database operations in the database specified in the transaction object named *Emp_tran*:

```
ROLLBACK USING emp_tran ;
```

SELECT

Description

Selects a row in the tables specified in *RestOfSelectStatement*.

Syntax

SELECT *RestOfSelectStatement*
    {USING *TransactionObject*} ;

Parameter	Description
RestOfSelectStatement	The rest of the SELECT statement (the column list INTO, FROM, WHERE, and other clauses).
TransactionObject	The name of the transaction object that identifies the database containing the table. This clause is required only for transaction objects other than the default (SQLCA).

Usage

An error occurs if the SELECT statement returns more than one row.

Error handling
It is good practice to test the success/failure code after executing a SELECT statement. You can test SQLCode for a failure code.

When you use the INTO clause, PocketBuilder does not verify whether the datatype of the retrieved column matches the datatype of the host variable; it only checks for the existence of the columns and tables. You are responsible for checking that the datatypes match. Keep in mind that not all database datatypes are the same as PocketBuilder datatypes.

Examples

The following statements select data in the Emp_LName and Emp_FName columns of a row in the Employee table and put the data into the SingleLineEdits sle_LName and sle_FName (the transaction object *Emp_tran* is used):

```
int     Emp_num
string   Emp_lname, Emp_fname
Emp_num = Integer(sle_Emp_Num.Text)

SELECT employee.Emp_LName, employee.Emp_FName
      INTO :Emp_lname, :Emp_fname
      FROM Employee
      WHERE Employee.Emp_nbr = :Emp_num
      USING Emp_tran ;

IF Emp_tran.SQLCode = 100 THEN
      MessageBox("Employee Inquiry", &
      "Employee Not Found")
ELSEIF Emp_tran.SQLCode > 0 then
      MessageBox("Database Error", &
      Emp_tran.SQLErrText, Exclamation!)
END IF
sle_Lname.text = Emp_lname
sle_Fname.text = Emp_fname
```

SELECTBLOB

Description

Selects a single blob column in a row in the table specified in *RestOfSelectStatement*.

Syntax

SELECTBLOB *RestOfSelectStatement*
{USING *TransactionObject*} ;

Parameter	Description
RestOfSelectStatement	The rest of the SELECT statement (the INTO, FROM, and WHERE clauses).
TransactionObject	The name of the transaction object that identifies the database containing the table. This clause is required only for transaction objects other than the default (SQLCA).

Usage

An error occurs if the SELECTBLOB statement returns more than one row.

Error handling
It is good practice to test the success/failure code after executing an SELECTBLOB statement. To make sure the update affected at least one row, check the SQLNRows property of SQLCA or the transaction object. The SQLCode or SQLDBCode property will not indicate the success or failure of the SELECTBLOB statement.

You can include an indicator variable in the host variable list (target parameters) in the INTO clause to check for an empty blob (a blob of zero length) and conversion errors.

Examples

The following statements select the blob column Emp_pic from a row in the Employee table and set the picture p_1 to the bitmap in *Emp_id_pic* (the transaction object *Emp_tran* is used):

```
Blob  Emp_id_pic
SELECTBLOB Emp_pic
       INTO  :Emp_id_pic
       FROM Employee
       WHERE Employee.Emp_Num = 100
       USING Emp_tran ;
p_1.SetPicture(Emp_id_pic)
```

The blob Emp_id_pic requires a colon to indicate that it is a host (PowerScript) variable when you use it in the INTO clause of the SELECTBLOB statement.

UPDATE

Description

Updates the rows specified in *RestOfUpdateStatement*.

Syntax

UPDATE *TableName RestOfUpdateStatement* {USING *TransactionObject*} ;

Parameter	Description
TableName	The name of the table in which you want to update rows.
RestOfUpdateStatement	The rest of the UPDATE statement (the SET and WHERE clauses).
TransactionObject	The name of the transaction object that identifies the database containing the table. This clause is required only for transaction objects other than the default (SQLCA).

Usage

Error handling
It is good practice to test the success/failure code after executing a UPDATE statement. You can test SQLCode for a failure code. However, if nothing matches the WHERE clause and no rows are updated, SQLCode is still set to zero. To make sure the update affected at least one row, check the SQLNRows property of the transaction object.

Examples

These statements update rows from the Employee table in the database specified in the transaction object named *Emp_tran*, where *Emp_num* is equal to the value entered in the SingleLineEdit sle_Number:

```
int Emp_num
Emp_num=Integer(sle_Number.Text )
UPDATE Employee
      SET emp_name = :sle_Name.Text
      WHERE Employee.emp_num  = :Emp_num
      USING Emp_tran ;

IF Emptran.SQLNRows > 0 THEN
      COMMIT USING Emp_tran ;
END IF
```

The integer *Emp_num* and the SingleLineEdit sle_name require a colon to indicate they are host (PowerScript) variables when you use them in an UPDATE statement.

UPDATEBLOB

Description

Updates the rows in *TableName* in *BlobColumn*.

Syntax

UPDATEBLOB *TableName*
    SET *BlobColumn* = *BlobVariable*
    RestOfUpdateStatement {USING *TransactionObject*} ;

Parameter	Description
TableName	The name of the table you want to update.
BlobColumn	The name of the column you want to update in *TableName*. The datatype of this column must be blob.
BlobVariable	A PowerScript variable of the datatype blob.

Parameter	Description
RestOfUpdateStatement	The rest of the UPDATE statement (the WHERE clause).
TransactionObject	The name of the transaction object that identifies the database containing the table. This clause is required only for transaction objects other than the default (SQLCA).

Usage

Error handling
It is good practice to test the success/failure code after executing an UPDATEBLOB statement. To make sure the update affected at least one row, check the SQLNRows property of SQLCA or the transaction object. The SQLCode or SQLDBCode property will not indicate the success or failure of the UPDATEBLOB statement.

Examples

These statements update the blob column emp_pic in the Employee table, where *emp_num* is 100:

```
int   fh
blob  Emp_id_pic
fh = FileOpen("c:\emp_100.bmp", StreamMode!)
IF fh <> -1 THEN
       FileRead(fh, emp_id_pic)
       FileClose(fh)
       UPDATEBLOB Employee SET emp_pic = :Emp_id_pic
       WHERE Emp_num = 100
       USING Emp_tran ;
END IF

IF Emptran.SQLNRows > 0 THEN
       COMMIT USING Emp_tran ;
END IF
```

The blob *Emp_id_pic* requires a colon to indicate it is a host (PowerScript) variable in the UPDATEBLOB statement.

UPDATE Where Current of Cursor

Description

Updates the row in which the cursor is positioned using the values in *SetStatement*.

Syntax

UPDATE *TableName SetStatement*
WHERE CURRENT OF *CursorName* ;

Parameter	Description
TableName	The name of the table in which you want to update the row
SetStatement	The word SET followed by a comma-separated list of the form *ColumnName* = *value*
CursorName	The name of the cursor in which the table is referenced

Usage

The USING Transaction Object clause is not allowed with UPDATE Where Current of Cursor; the transaction object was specified in the statement that declared the cursor.

Examples

This statement updates the row in the Employee table in which the cursor called *Emp_curs* is positioned:

```
UPDATE Employee
      SET salary = 17800
      WHERE CURRENT of Emp_curs ;
```

Using dynamic SQL

General information

Because database applications usually perform a specific activity, you usually know the complete SQL statement when you write and compile the script. When PocketBuilder does not support the statement in embedded SQL (as with a DDL statement) or when the parameters or the format of the statements are unknown at compile time, the application must build the SQL statements at execution time. This is called dynamic SQL. The parameters used in dynamic SQL statements can change each time the program is executed.

Using Adaptive Server® Anywhere
For information about using dynamic SQL with Adaptive Server Anywhere, see the Adaptive Server Anywhere *Programming Interfaces* book.

Four formats

PocketBuilder has four dynamic SQL formats. Each format handles one of the following situations at compile time:

Format	When used
Format 1	Non-result-set statements with no input parameters
Format 2	Non-result-set statements with input parameters
Format 3	Result-set statements in which the input parameters and result-set columns are known at compile time
Format 4	Result-set statements in which the input parameters, the result-set columns or both are unknown at compile time

To handle these situations, you use:

- The PocketBuilder dynamic SQL statements
- The dynamic versions of CLOSE, DECLARE, FETCH, OPEN, and EXECUTE
- The PocketBuilder datatypes DynamicStagingArea and DynamicDescriptionArea

About the examples
The examples assume that the default transaction object (SQLCA) has been assigned valid values and that a successful CONNECT has been executed. Although the examples do not show error checking, you should check the SQLCode after each SQL statement.

Dynamic SQL statements

The PocketBuilder dynamic SQL statements are:

```
DESCRIBE DynamicStagingArea
     INTO DynamicDescriptionArea ;

EXECUTE {IMMEDIATE} SQLStatement
     {USING TransactionObject} ;

EXECUTE DynamicStagingArea
     USING ParameterList ;

EXECUTE DYNAMIC Cursor | Procedure
     USING ParameterList ;

OPEN DYNAMIC Cursor | Procedure
     USING ParameterList ;

EXECUTE DYNAMIC Cursor | Procedure
     USING DESCRIPTOR DynamicDescriptionArea ;
```

```
OPEN DYNAMIC Cursor | Procedure
    USING DESCRIPTOR DynamicDescriptionArea ;

PREPARE DynamicStagingArea
    FROM SQLStatement {USING TransactionObject} ;
```

Two datatypes

DynamicStagingArea DynamicStagingArea is a PowerBuilder datatype. PowerBuilder uses a variable of this type to store information for use in subsequent statements.

The DynamicStagingArea is the only connection between the execution of a statement and a transaction object and is used internally by PowerBuilder; you cannot access information in the DynamicStagingArea.

PowerBuilder provides a global DynamicStagingArea variable named SQLSA that you can use when you need a DynamicStagingArea variable.

If necessary, you can declare and create additional object variables of the type DynamicStagingArea. These statements declare and create the variable, which must be done before referring to it in a dynamic SQL statement:

```
DynamicStagingArea dsa_stage1
dsa_stage1 = CREATE DynamicStagingArea
```

After the EXECUTE statement is completed, SQLSA is no longer referenced.

DynamicDescriptionArea DynamicDescriptionArea is a PowerBuilder datatype. PowerBuilder uses a variable of this type to store information about the input and output parameters used in Format 4 of dynamic SQL.

PowerBuilder provides a global DynamicDescriptionArea named SQLDA that you can use when you need a DynamicDescriptionArea variable.

If necessary, you can declare and create additional object variables of the type DynamicDescriptionArea. These statements declare and create the variable, which must be done before referring to it in a dynamic SQL statement:

```
DynamicDescriptionArea dda_desc1
dsa_desc1 = CREATE DynamicDescriptionArea
```

For more information about SQLDA, see Dynamic SQL Format 4 on page 165.

Preparing to use dynamic SQL

When you use dynamic SQL, you must:

- Prepare the DynamicStagingArea in all formats except Format 1
- Describe the DynamicDescriptionArea in Format 4
- Execute the statements in the appropriate order

Preparing and describing the datatypes Since the SQLSA staging area is the only connection between the execution of a SQL statement and a transaction object, an execution error will occur if you do not prepare the SQL statement correctly.

In addition to SQLSA and SQLDA, you can declare other variables of the DynamicStagingArea and DynamicDescriptionArea datatypes. However, this is required only when your script requires simultaneous access to two or more dynamically prepared statements.

This is a *valid* dynamic cursor:

```
DECLARE my_cursor DYNAMIC CURSOR FOR SQLSA ;
PREPARE SQLSA FROM "SELECT emp_id FROM employee" ;
OPEN DYNAMIC my_cursor ;
```

This is an *invalid* dynamic cursor. There is no PREPARE, and therefore an execution error will occur:

```
DECLARE my_cursor DYNAMIC CURSOR FOR SQLSA ;
OPEN DYNAMIC my_cursor ;
```

Statement order Where you place the dynamic SQL statements in your scripts is unimportant, but the order of execution is important in Formats 2, 3, and 4. You must execute:

1. The DECLARE and the PREPARE before you execute any other dynamic SQL statements
2. The OPEN in Formats 3 and 4 before the FETCH
3. The CLOSE at the end

If you have multiple PREPARE statements, the order affects the contents of SQLSA.

These statements illustrate the correct ordering:

```
DECLARE my_cursor DYNAMIC CURSOR FOR SQLSA
string sql1, sql2
sql1 = "SELECT emp_id FROM department "&
WHERE salary > 90000"
sql2 = "SELECT emp_id FROM department "&
WHERE salary > 20000"

IF deptId = 200 then
      PREPARE SQLSA FROM :sql1 USING SQLCA ;
```

```
ELSE
      PREPARE SQLSA FROM :sql2 USING SQLCA ;
END IF
OPEN DYNAMIC my_cursor ;      // my_cursor maps to the
                              // SELECT that has been
                              // prepared.
```

Dynamic SQL Format 1

Description

Use this format to execute a SQL statement that does not produce a result set and does not require input parameters. You can use this format to execute all forms of Data Definition Language (DDL).

Syntax

EXECUTE IMMEDIATE *SQLStatement*
{USING *TransactionObject*} ;

Parameter	Description
SQLStatement	A string containing a valid SQL statement. The string can be a string constant or a PowerBuilder variable preceded by a colon (such as :mysql). The string must be contained on one line and cannot contain expressions.
TransactionObject (optional)	The name of the transaction object that identifies the database.

Examples

These statements create a database table named Trainees. The statements use the string *Mysql* to store the CREATE statement.

```
string  Mysql
Mysql = "CREATE TABLE Trainees "&
      +"(emp_id integer not null,"&
      +"emp_fname char(10) not null, "&
      +"emp_lname char(20) not null)"
EXECUTE IMMEDIATE :Mysql ;
```

These statements assume a transaction object named *My_trans* exists and is connected:

```
string  Mysql
Mysql="INSERT INTO department Values (1234,"&
       +"'Purchasing', 1234)"
EXECUTE IMMEDIATE :Mysql USING My_trans ;
```

Dynamic SQL Format 2

Description

Use this format to execute a SQL statement that does not produce a result set but does require input parameters. You can use this format to execute all forms of Data Definition Language (DDL).

Syntax

PREPARE *DynamicStagingArea* FROM *SQLStatement*
{USING *TransactionObject*} ;

EXECUTE *DynamicStagingArea* USING {*ParameterList*} ;

Parameter	Description
DynamicStagingArea	The name of the DynamicStagingArea (usually SQLSA). If you need a DynamicStagingArea variable other than SQLSA, you must declare it and instantiate it with the CREATE statement before using it.
SQLStatement	A string containing a valid SQL statement. The string can be a string constant or a PowerBuilder variable preceded by a colon (such as :mysql). The string must be contained on one line and cannot contain expressions. Enter a question mark (?) for each parameter in the statement. Value substitution is positional; reserved word substitution is not allowed.
TransactionObject (optional)	The name of the transaction object that identifies the database.
ParameterList (optional)	A comma-separated list of PowerScript variables. Note that PowerScript variables are preceded by a colon (:).

Usage

To specify a null value, use the SetNull function.

Examples

These statements prepare a DELETE statement with one parameter in SQLSA and then execute it using the value of the PowerScript variable *Emp_id_var*:

```
INT  Emp_id_var = 56
PREPARE SQLSA
      FROM "DELETE FROM employee WHERE emp_id=?" ;
EXECUTE SQLSA USING :Emp_id_var ;
```

These statements prepare an INSERT statement with three parameters in SQLSA and then execute it using the value of the PowerScript variables *Dept_id_var*, *Dept_name_var*, and *Mgr_id_var* (note that *Mgr_id_var* is null):

```
INT Dept_id_var = 156
INT Mgr_id_var
String  Dept_name_var
Dept_name_var = "Department"
SetNull(Mgr_id_var)
```

```
PREPARE SQLSA
      FROM "INSERT INTO department VALUES (?,?,?)" ;
EXECUTE SQLSA
       USING :Dept_id_var,:Dept_name_var,:Mgr_id_var ;
```

Dynamic SQL Format 3

Description

Use this format to execute a SQL statement that produces a result set in which the input parameters and result set columns are known at compile time.

Syntax

DECLARE *Cursor* | *Procedure*
DYNAMIC CURSOR | PROCEDURE
FOR *DynamicStagingArea* ;

PREPARE *DynamicStagingArea* FROM *SQLStatement*
{USING *TransactionObject*} ;

OPEN DYNAMIC *Cursor*
{USING *ParameterList*} ;

EXECUTE DYNAMIC *Procedure*
{USING *ParameterList*} ;

FETCH *Cursor* | *Procedure*
INTO *HostVariableList* ;

CLOSE *Cursor* | *Procedure* ;

Parameter	**Description**
Cursor or *Procedure*	The name of the cursor or procedure you want to use.
DynamicStagingArea	The name of the DynamicStagingArea (usually SQLSA). If you need a DynamicStagingArea variable other than SQLSA, you must declare it and instantiate it with the CREATE statement before using it.
SQLStatement	A string containing a valid SQL SELECT statement The string can be a string constant or a PowerBuilder variable preceded by a colon (such as :mysql). The string must be contained on one line and cannot contain expressions. Enter a question mark (?) for each parameter in the statement. Value substitution is positional; reserved word substitution is not allowed.
TransactionObject (optional)	The name of the transaction object that identifies the database.

Parameter	Description
ParameterList (optional)	A comma-separated list of PowerScript variables. Note that PowerScript variables are preceded by a colon (:).
HostVariableList	The list of PowerScript variables into which the data values will be retrieved.

Usage

To specify a null value, use the SetNull function.

The DECLARE statement is not executable and can be declared globally.

If your DBMS supports formats of FETCH other than the customary (and default) FETCH NEXT, you can specify FETCH FIRST, FETCH PRIOR, or FETCH LAST.

The FETCH and CLOSE statements in Format 3 are the same as in standard embedded SQL.

To declare a local cursor or procedure, open the script in the Script view and select Paste SQL from the PainterBar or the Edit>Paste Special menu. To declare a global, instance, or shared cursor or procedure, select Declare from the first drop-down list in the Script view, and select Global Variables, Instance Variables, or Shared Variables from the second drop-down list. Then, select Paste SQL.

For information about global, instance, shared, and local scope, see “Where to declare variables” on page 31.

Examples

Example 1 These statements associate a cursor named *my_cursor* with SQLSA, prepare a SELECT statement in SQLSA, open the cursor, and return the employee ID in the current row into the PowerScript variable *Emp_id_var*:

```
integer Emp_id_var
DECLARE my_cursor DYNAMIC CURSOR FOR SQLSA ;
PREPARE SQLSA FROM "SELECT emp_id FROM employee" ;
OPEN DYNAMIC my_cursor ;
FETCH my_cursor INTO :Emp_id_var ;
CLOSE my_cursor ;
```

You can loop through the cursor as you can in embedded static SQL.

Example 2 These statements associate a cursor named *my_cursor* with SQLSA, prepare a SELECT statement with one parameter in SQLSA, open the cursor, and substitute the value of the variable *Emp_state_var* for the parameter in the SELECT statement. The employee ID in the active row is returned into the PowerBuilder variable *Emp_id_var*:

```
DECLARE my_cursor DYNAMIC CURSOR FOR SQLSA ;
integer Emp_id_var
```

```
string Emp_state_var = "MA"
string sqlstatement

sqlstatement = "SELECT emp_id FROM employee "&
      +"WHERE state = ?"
PREPARE SQLSA FROM :sqlstatement ;
OPEN DYNAMIC my_cursor using :Emp_state_var ;
FETCH my_cursor INTO :Emp_id_var ;
CLOSE my_cursor ;
```

Example 3 These statements perform the same processing as the preceding example but use a database stored procedure called *Emp_select*:

```
// The syntax of emp_select is:
// create procedure emp_select (@stateparm char(2)) as
// SELECT emp_id FROM employee WHERE state=@stateparm.
DECLARE my_proc DYNAMIC PROCEDURE FOR SQLSA ;
integer Emp_id_var
string Emp_state_var

PREPARE SQLSA FROM "execute emp_select @stateparm=?" ;
Emp_state_var = "MA"
EXECUTE DYNAMIC my_proc USING :Emp_state_var ;
FETCH my_proc INTO :Emp_id_var ;
CLOSE my_proc ;
```

Dynamic SQL Format 4

Description

Use this format to execute a SQL statement that produces a result set in which the number of input parameters, or the number of result-set columns, or both, are unknown at compile time.

Syntax

```
DECLARE Cursor | Procedure
      DYNAMIC CURSOR | PROCEDURE
      FOR DynamicStagingArea ;

PREPARE DynamicStagingArea FROM SQLStatement
      {USING TransactionObject} ;

DESCRIBE DynamicStagingArea
      INTO DynamicDescriptionArea ;

OPEN DYNAMIC Cursor | Procedure
      USING DESCRIPTOR DynamicDescriptionArea ;

EXECUTE DYNAMIC Cursor | Procedure
      USING DESCRIPTOR DynamicDescriptionArea ;
```

```
FETCH Cursor | Procedure
          USING DESCRIPTOR DynamicDescriptionArea ;
```

```
CLOSE Cursor | Procedure ;
```

Parameter	**Description**
Cursor or *Procedure*	The name of the cursor or procedure you want to use.
DynamicStagingArea	The name of the DynamicStagingArea (usually SQLSA). If you need a DynamicStagingArea variable other than SQLSA, you must declare it and instantiate it with the CREATE statement before using it.
SQLStatement	A string containing a valid SQL SELECT statement. The string can be a string constant or a PowerBuilder variable preceded by a colon (such as :*mysql*). The string must be contained on one line and cannot contain expressions. Enter a question mark (?) for each parameter in the statement. Value substitution is positional; reserved word substitution is not allowed.
TransactionObject (optional)	The name of the transaction object that identifies the database.
DynamicDescriptionArea	The name of the DynamicDescriptionArea (usually SQLDA). If you need a DynamicDescriptionArea variable other than SQLDA, you must declare it and instantiate it with the CREATE statement before using it.

Usage

The DECLARE statement is not executable and can be defined globally.

If your DBMS supports formats of FETCH other than the customary (and default) FETCH NEXT, you can specify FETCH FIRST, FETCH PRIOR, or FETCH LAST.

To declare a local cursor or procedure, open the script in the Script view and select Paste SQL from the PainterBar or the Edit>Paste Special menu. To declare a global, instance, or shared cursor or procedure, select Declare from the first drop-down list in the Script view and Global Variables, Instance Variables, or Shared Variables from the second drop-down list, then select Paste SQL.

For information about global, instance, shared, and local scope, see “Where to declare variables” on page 31.

Accessing attribute information When a statement is described into a DynamicDescriptionArea, this information is available to you in the attributes of that DynamicDescriptionArea variable:

Information	Attribute
Number of input parameters	NumInputs
Array of input parameter types	InParmType
Number of output parameters	NumOutputs
Array of output parameter types	OutParmType

Setting and accessing parameter values The array of input parameter values and the array of output parameter values are also available. You can use the SetDynamicParm function to set the values of an input parameter and the following functions to obtain the value of an output parameter:

GetDynamicDate
GetDynamicDateTime
GetDynamicNumber
GetDynamicString
GetDynamicTime

For information about these functions, see GetDynamicDate on page 528, GetDynamicDateTime on page 528, GetDynamicNumber on page 529, GetDynamicString on page 529, and GetDynamicTime on page 529.

Parameter values The following enumerated datatypes are the valid values for the input and output parameter types:

TypeBoolean!
TypeDate!
TypeDateTime!
TypeDecimal!
TypeDouble!
TypeInteger!
TypeLong!
TypeReal!
TypeString!
TypeTime!
TypeUInt!
TypeULong!
TypeUnknown!

Input parameters You can set the type and value of each input parameter found in the PREPARE statement. PowerBuilder populates the SQLDA attribute NumInputs when the DESCRIBE is executed. You can use this value with the SetDynamicParm function to set the type and value of a specific input parameter. The input parameters are optional; but if you use them, you should fill in all the values before executing the OPEN or EXECUTE statement.

Output parameters You can access the type and value of each output parameter found in the PREPARE statement. If the database supports output parameter description, PowerBuilder populates the SQLDA attribute NumOutputs when the DESCRIBE is executed. If the database does not support output parameter description, PowerBuilder populates the SQLDA attribute NumOutputs when the FETCH statement is executed.

You can use the number of output parameters in the NumOutputs attribute in functions to obtain the type of a specific parameter from the output parameter type array in the OutParmType attribute. When you have the type, you can call the appropriate function after the FETCH statement to retrieve the output value.

Examples

Example 1 These statements assume you know that there will be only one output descriptor and that it will be an integer. You can expand this example to support any number of output descriptors and any datatype by wrapping the CHOOSE CASE statement in a loop and expanding the CASE statements:

```
string Stringvar, Sqlstatement
integer Intvar
Long LongVar

Sqlstatement = "SELECT emp_id FROM employee"
PREPARE SQLSA FROM :Sqlstatement ;
DESCRIBE SQLSA INTO SQLDA ;
DECLARE my_cursor DYNAMIC CURSOR FOR SQLSA ;
OPEN DYNAMIC my_cursor USING DESCRIPTOR SQLDA ;
FETCH my_cursor USING DESCRIPTOR SQLDA ;
// If the FETCH is successful, the output
// descriptor array will contain returned
// values from the first row of the result set.
// SQLDA.NumOutputs contains the number of
// output descriptors.
// The SQLDA.OutParmType array will contain
// NumOutput entries and each entry will contain
// a value of the enumerated datatype ParmType
// (such as TypeInteger!, TypeLong!, or TypeString!).
CHOOSE CASE SQLDA.OutParmType[1]
      CASE TypeString!
         Stringvar = GetDynamicString(SQLDA, 1)
```

```
        CASE TypeInteger!
           Intvar = GetDynamicNumber(SQLDA, 1)
        CASE TypeLong!
           Longvar = GetDynamicNumber(SQLDA, 1)
END CHOOSE
CLOSE my_cursor ;
```

Example 2 These statements assume you know there is one string input descriptor and sets the parameter to MA:

```
string Sqlstatement, sValue
Sqlstatement = "SELECT emp_fname, emp_lname " &
       + "FROM employee WHERE state = ?"
PREPARE SQLSA FROM :Sqlstatement ;

DESCRIBE SQLSA INTO SQLDA ;

// If the DESCRIBE is successful, the input
// descriptor array will contain one input
// descriptor that you must fill prior to the OPEN

DECLARE my_cursor DYNAMIC CURSOR FOR SQLSA ;
SetDynamicParm(SQLDA, 1, "MA")

OPEN DYNAMIC my_cursor USING DESCRIPTOR SQLDA ;

FETCH my_cursor USING DESCRIPTOR SQLDA ;

// If the FETCH is successful, the output
// descriptor array will contain returned
// values from the first row of the result set
// as in the first example.

// To test and see the values:
sValue = SQLDA.GetDynamicString(1)
//messagebox("",sValue)
sValue = SQLDA.GetDynamicString(2)
//messagebox("",sValue)
Do While sqlca.sqlcode <> 100
   FETCH my_cursor USING DESCRIPTOR SQLDA ;
      sValue = SQLDA.GetDynamicString(1)
      //messagebox("",sValue)
      sValue = SQLDA.GetDynamicString(2)
     //messagebox("",sValue)
Loop

CLOSE my_cursor ;
```

CHAPTER 9

PowerScript Events

About this chapter

This chapter discusses events in general and then documents the arguments, event IDs, and return codes for the events defined for all PocketBuilder controls and objects except the DataWindow and DataStore. Usage notes and examples provide information about what is typically done in an event's script.

For information about DataWindow and DataStore events, see the *DataWindow Reference*.

Contents

The events are listed in alphabetical order.

About events

In PocketBuilder, there are several types of events.

Table 9-1: PocketBuilder event types

Type	Occurs in response to
System events with an ID	User actions or other system messages or a call in your scripts
System events without an ID	PocketBuilder messages or a call in your scripts
User-defined events with an ID	User actions or other system messages or a call in your scripts
User-defined events without an ID	A call in your scripts

The following information about event IDs, arguments, and return values applies to all types of events.

Event IDs

An event ID connects an event to a system message. Events that can be triggered by user actions or other system activity have event IDs. In PocketBuilder's objects, PocketBuilder defines events for commonly used event IDs. These events are documented in this chapter. You can define your own events for other system messages using the event IDs listed in the Event Declaration dialog box.

Events without IDs Some system events, such as the application object's Open event, do not have an event ID. They are associated with PocketBuilder activity, not system activity. PocketBuilder triggers them itself when appropriate.

Arguments

System-triggered events Each system event has its own list of zero or more arguments. When PocketBuilder triggers the event in response to a system message, it supplies values for the arguments, which become available in the event script.

Events you trigger If you trigger a system event in another event script, you specify the expected arguments. For example, in the Clicked event for a window, you can trigger the DoubleClicked event with this statement, passing its flags, xpos, and ypos arguments on to the DoubleClicked event.

```
w_main.EVENT DoubleClicked(flags, xpos, ypos)
```

Because DoubleClicked is a system event, the argument list is fixed—you cannot supply additional arguments of your own.

Calling events without specifying their arguments
If you use the CALL statement, you can trigger a system event without specifying its arguments. However, CALL is obsolete and you should not use it in new applications except as described in CALL on page 116.

Return values

Where does the return value go? Most events have a return value. When the event is triggered by the system, the return value is returned to the system.

When your script triggers a user-defined or system event, you can capture the return value in an assignment statement:

```
li_rtn = w_main.EVENT process_info(mydata)
```

When you post an event, the return value is lost because the calling script is no longer running when the posted script is actually run. The compiler does not allow a posted event in an assignment statement.

Return codes System events with return values have a default return code of 0, which means, "take no special action and continue processing." Some events have additional codes that you can return to change the processing that happens after the event. For example, a return code might allow you to suppress an error message or prevent a change from taking place.

A RETURN statement is not required in an event script, but for most events it is good practice to include one. For events with return values, if you do not have a RETURN statement, the event returns 0.

Some system events have no return value. For these events, the compiler does not allow a RETURN statement.

Ancestor event script return values

Sometimes you want to perform some processing in an event in a descendent object, but that processing depends on the return value of the ancestor event script. You can use a local variable called *AncestorReturnValue* that is automatically declared and assigned the value of the ancestor event.

For more information about *AncestorReturnValue*, see “Calling functions and events in an object’s ancestor” on page 108.

User-defined events

With an ID When you declare a user-defined event that will be triggered by a system message, you select an event ID from the list of IDs. The pbm (PowerBuilder Message) codes listed in the Event dialog box map to system messages.

The return value and arguments associated with the event ID become part of your event declaration. You cannot modify them.

When the corresponding system message occurs, PocketBuilder triggers the event and passes values for the arguments to the event script.

Without an ID When you declare a user event that will not be associated with a system message, you do not select an event ID for the event.

You can specify your own arguments and return datatype in the Event Declaration dialog box.

The event will never be triggered by user actions or system activity. You trigger the event yourself in your application’s scripts.

For more information

If you want to trigger events, including system events, see “Syntax for calling PocketBuilder functions and events” on page 104 for information on the calling syntax.

To learn more about user-defined events, see the *Users Guide*.

Activate

Description

Occurs just before the window becomes active.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Event ID

Event ID	Objects
pbm_activate	Window

Arguments

None

Return codes

Long. Return code choices (specify in a RETURN statement):

0 Continue processing

Usage

When an Activate event occurs, the first object in the tab order for the window gets focus. If there are no visible objects in the window, the window gets focus.

An Activate event occurs for a newly opened window because it is made active after it is opened.

The Activate event is frequently used to enable and disable menu items.

Examples

Example 1 In the window's Activate event, this code disables the Sheet menu item for menu m_frame on the File menu:

```
m_frame.m_file.m_sheet.Enabled = FALSE
```

Example 2 This code opens the sheet w_sheet in a layered style when the window activates:

```
w_sheet.ArrangeSheets(Layer!)
```

See also

Close
Open
Show

BeginDownload

Description

Reserved for future use. Occurs at the beginning of a download procedure

Event ID

Event ID	Objects
None	MLSynchronization, MLSync

Arguments

None

Return codes

None

BeginDrag

The BeginDrag event has different arguments for different objects:

Object	See
ListView control	Syntax 1
TreeView control	Syntax 2

Syntax 1 For ListView controls

Description

Occurs when the user presses the left mouse button in the ListView control and begins dragging.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✓

Event ID

Event ID	Objects
pbm_lvnbegindrag	ListView

Arguments

Argument	Description
index	Integer by value (the index of the ListView item being dragged)

Return codes

Long. Return code choices (specify in a RETURN statement):

0 Continue processing

Usage

BeginDrag and BeginRightDrag events occur when the user presses the mouse button and drags, whether or not dragging is enabled. To enable dragging, you can:

- Set the DragAuto property to true. If the ListView's DragAuto property is true, a drag operation begins automatically when the user clicks.
- Call the Drag function. If DragAuto is false, then in the BeginDrag event script, the programmer can call the Drag function to begin the drag operation.

Dragging a ListView item onto another control causes its standard drag events (DragDrop, DragEnter, DragLeave, and DragWithin) to occur. The standard drag events occur for ListView when another control is dragged within the borders of the ListView.

Examples

This example moves a ListView item from one ListView to another. *Ilvi_dragged_object* is a window instance variable whose type is ListViewItem. To copy the item, omit the code that deletes it from the source ListView.

This code is in the BeginDrag event script of the source ListView:

```
// If the TreeView's DragAuto property is FALSE
This.Drag(Begin!)

This.GetItem(This.SelectedIndex(), &
   ilvi_dragged_object)

// To copy, rather than move, omit these two lines
This.DeleteItem(This.SelectedIndex())
This.Arrange()
```

This code is in the DragDrop event of the target ListView:

```
This.AddItem(ilvi_dragged_object)
This.Arrange()
```

See also

BeginRightDrag
DragDrop
DragEnter
DragLeave
DragWithin

Syntax 2 For TreeView controls

Description

Occurs when the user presses the left mouse button on a label in the TreeView control and begins dragging.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Event ID

Event ID	Objects
pbm_tvnbegindrag	TreeView

Arguments

Argument	Description
handle	Long by value (handle of the TreeView item being dragged)

Return codes

Long. Return code choices (specify in a RETURN statement):

0 Continue processing

Usage

BeginDrag and BeginRightDrag events occur when the user presses the mouse button and drags, whether or not dragging is enabled. To enable dragging, you can:

- Set the DragAuto property to true. If the TreeView's DragAuto property is true, a drag operation begins automatically when the user clicks.
- Call the Drag function. If DragAuto is false, then in the BeginDrag event script, the programmer can call the Drag function to begin the drag operation.

The user cannot drag a highlighted item.

Dragging a TreeView item onto another control causes the control's standard drag events (DragDrop, DragEnter, DragLeave, and DragWithin) to occur. The standard drag events occur for TreeView when another control is dragged within the borders of the TreeView.

Examples

This example moves the first TreeView item in the source TreeView to another TreeView when the user drags there. *Itvi_dragged_object* is a window instance variable whose type is TreeViewItem. To copy the item, omit the code that deletes it from the source TreeView.

This code is in the BeginDrag event script of the source TreeView:

```
long itemnum
```

```
// If the TreeView's DragAuto property is FALSE
This.Drag(Begin!)
itemnum = 1
This.GetItem(itemnum, itvi_dragged_object)

// To copy, rather than move, omit these two lines
This.DeleteItem(itemnum)
This.SetRedraw(TRUE)
```

This code is in the DragDrop event of the target TreeView:

```
This.InsertItemLast(0, ilvi_dragged_object)
This.SetRedraw(TRUE)
```

Instead of deleting the item from the source TreeView immediately, consider deleting it after the insertion in the DragDrop event succeeds.

See also

BeginRightDrag
DragDrop
DragEnter
DragLeave
DragWithin

BeginLabelEdit

The BeginLabelEdit event has different arguments for different objects:

Object	See
ListView control	Syntax 1
TreeView control	Syntax 2

Syntax 1 For ListView controls

Description

Occurs when the user clicks on the label of an item after selecting the item.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Event ID

Event ID	Objects
pbm_lvnbeginlabeledit	ListView

Arguments

Argument	Description
index	Integer by value (the index of the selected ListView item)

Return codes

Long. Return code choices (specify in a RETURN statement):

0	Allow editing of the label
1	Prevent editing of the label

Usage

When editing is allowed, a box appears around the label with the text highlighted. The user can replace or change the existing text.

Examples

This example uses the BeginLabelEdit event to display the name of the ListView item being edited:

```
ListViewItem lvi
This.GetItem(index lvi)
sle_info.text = "Editing " + string(lvi.label)
```

See also

EndLabelEdit

Syntax 2 For TreeView controls

Description

Occurs when the user clicks on the label of an item after selecting the item.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Event ID

Event ID	Objects
pbm_tvnbeginlabeledit	TreeView

Arguments

Argument	Description
handle	Long by value (the handle of the selected TreeView item)

Return codes

Long. Return code choices (specify in a RETURN statement):

0	Allow editing of the label
1	Prevent editing of the label

Usage

When editing is allowed, a box appears around the label with the text highlighted. The user can replace or change the existing text.

Examples

This example uses the BeginLabelEdit to display the name of the TreeView item being edited in a SingleLineEdit:

```
TreeViewItem tvi
This.GetItem(index, tvi)
sle_info.text = "Editing " + string(tvi.label)
```

See also

EndLabelEdit

BeginLogScan

Description

Reserved for future use. Occurs before dbmlsync scans the transaction log to assemble the upload data stream.

Event ID

Event ID	Objects
None	MLSync

Arguments

Argument	Description
rescanlog	Boolean indicating whether the log has already been scanned for the current synchronization.

Return codes

None

BeginRightDrag

The BeginRightDrag event has different arguments for different objects:

Object	See
ListView control	Syntax 1
TreeView control	Syntax 2

Syntax 1 For ListView controls

Description

Occurs when the user presses the right mouse button in the ListView control and begins dragging.

PocketBuilder	✗
PowerBuilder	✓

Syntax 2 For TreeView controls

Description

Occurs when the user presses the right mouse button in the TreeView control and begins dragging.

PocketBuilder	✗
PowerBuilder	✓

BeginSync

Description

Reserved for future use. Occurs at the beginning of the synchronization.

Event ID

Event ID	**Objects**
None	MLSynchronization, MLSync

Arguments

Argument	**Description**
mlusername	Read-only string identifying the MobiLink user name.
pubnames	Read-only string identifying the publication to be synchronized. If there is more than one publication, this is a comma-separated list.

Return codes

None

BeginUpload

Description

Reserved for future use. Occurs at the beginning of the synchonization upload procedure.

Event ID

Event ID	Objects
None	MLSynchronization, MLSync

Arguments

None

Return codes

None

Clicked

The Clicked event has different arguments for different objects:

Object	See
Menus	Syntax 1
ListView and Toolbar controls	Syntax 2
Tab controls	Syntax 3
TreeView controls	Syntax 4
Window	Syntax 5
Other controls	Syntax 6

For information about the DataWindow control’s Clicked event, see the *DataWindow Reference* or the online Help.

Syntax 1 For menus

Description

Occurs when the user chooses an item on a menu.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Event ID

Event ID	Objects
None	Menu

Arguments	None
Return codes	None (do not use a RETURN statement)

Usage

If the user highlights the menu item without choosing it, its Selected event occurs.

If the user chooses a menu item that has a cascaded menu associated with it, the Clicked event occurs, and the cascaded menu is displayed.

Examples

This script is for the Clicked event of the New menu item for the frame window. The wf_newsheet function is a window function. The window w_genapp_frame is part of the application template you can generate when you create a new application:

```
/* Create a new sheet */
w_genapp_frame.wf_newsheet( )
```

See also

Selected

Syntax 2 For ListView controls

Description

Occurs when the user clicks within the ListView or Toolbar control, either on an item or in the blank space around items.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Event ID

Event ID	Objects
pbm_lvnclicked	ListView

Arguments

Argument	Description
index	Integer by value (the index of the ListView item the user clicked). The value of *index* is -1 if the user clicks within the control but not on a specific item.

Return codes

Long. Return code choices (specify in a RETURN statement):

0 Continue processing

Usage

The Clicked event occurs when the user presses the mouse button. The Clicked event can occur during a double-click, in addition to the DoubleClicked event.

In addition to the Clicked event, ItemChanging and ItemChanged events can occur when the user clicks on an item that does not already have focus. BeginLabelEdit can occur when the user clicks on a label of an item that has focus.

Using the ItemActivate event for ListView controls
You can use the ItemActivate event (with the OneClickActivate property set to true) instead of the Clicked event for ListView controls.

Examples

This code changes the label of the item the user clicks to uppercase:

```
IF index = -1 THEN RETURN 0

This.GetItem(index, llvi_current)
llvi_current.Label = Upper(llvi_current.Label)
This.SetItem(index, llvi_current)
RETURN 0
```

See also

ColumnClick
DoubleClicked
ItemActivate
ItemChanged
ItemChanging
RightClicked
RightDoubleClicked

Syntax 3 For Tab controls

Description

Occurs when the user clicks on the tab portion of a Tab control.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✓

Event ID

Event ID	Objects
pbm_tcnclicked	Tab

Arguments

Argument	Description
index	Integer by value (the index of the tab page the user clicked)

Return codes

Long. Return code choices (specify in a RETURN statement):

0	Continue processing

Usage

The Clicked event occurs when the mouse button is released.

When the user clicks in the display area of the Tab control, the tab page user object (not the Tab control) gets a Clicked event.

The Clicked event can occur during a double-click, in addition to the DoubleClicked event.

In addition to the Clicked event, the SelectionChanging and SelectionChanged events can occur when the user clicks on a tab page label. If the user presses an arrow key to change tab pages, the Key event occurs instead of Clicked before SelectionChanging and SelectionChanged.

Examples

This code makes the tab label bold for the fourth tab page only:

```
IF index = 4 THEN
   This.BoldSelectedText = TRUE
ELSE
   This.BoldSelectedText = FALSE
END IF
```

See also

DoubleClicked
RightClicked
RightDoubleClicked
SelectionChanged
SelectionChanging

Syntax 4 For TreeView controls

Description

Occurs when the user clicks an item in a TreeView control.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Event ID

Event ID	Objects
pbm_tvnclicked	TreeView

Arguments

Argument	Description
handle	Long by value (the handle of the TreeView item the user clicked)

Return codes

Long. Return code choices (specify in a RETURN statement):

0 Continue processing

Usage

The Clicked event occurs when the user presses the mouse button.

The Clicked event can occur during a double-click, in addition to the DoubleClicked event.

In addition to the Clicked event, GetFocus occurs if the control does not already have focus.

Examples

This code in the Clicked event changes the label of the item the user clicked to uppercase:

```
TreeViewItem ltvi_current

This.GetItem(handle, ltvi_current)
ltvi_current.Label = Upper(ltvi_current.Label)
This.SetItem(handle, ltvi_current)
```

See also

DoubleClicked
RightClicked
RightDoubleClicked
SelectionChanged
SelectionChanging

Syntax 5 For windows

Description

Occurs when the user clicks in an unoccupied area of the window (any area with no visible, enabled object).

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Event ID

Event ID	Objects
pbm_lbuttonclk	Window

Arguments

Argument	Description
flags	UnsignedLong by value (the modifier keys and mouse buttons that are pressed). Values are: • 1 — Left mouse button • 2 — Right mouse button • 4 — Shift key • 8 — Ctrl key • 16 — Middle mouse button In the Clicked event, the left mouse button is being released, so 1 is not summed in the value of *flags*. For an explanation of *flags*, see Syntax 2 of MouseMove on page 245.
xpos	Integer by value (the distance of the pointer from the left edge of the window's workspace in pixels).
ypos	Integer by value (the distance of the pointer from the top of the window's workspace in pixels).

Return codes

Long. Return code choices (specify in a RETURN statement):

0 Continue processing

Usage

The Clicked event occurs when the user releases the mouse button.

If the user clicks on a control or menu, that object (rather than the window) gets a Clicked event. No Clicked event occurs when the user clicks the window's title bar.

When the user clicks on the window, the window's MouseDown and MouseUp events also occur.

When the user clicks on a visible disabled control or an invisible enabled control, the window gets a Clicked event.

Examples

If the user clicks in the upper left corner of the window, this code sets focus to the button cb_clear:

```
IF (xpos <= 600 AND ypos <= 600) THEN
   cb_clear.SetFocus( )
END IF
```

See also

DoubleClicked
MouseDown
MouseMove
MouseUp
RButtonDown

Syntax 6 For other controls

Description

Occurs when the user clicks on the control.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Event ID

Event ID	Objects
pbm_bnclicked	CheckBox, CommandButton, Graph, Picture, PictureHyperLink, PictureButton, RadioButton, StaticText, StaticHyperLink
pbm_prnclicked	HProgressBar, VProgressBar

Arguments

None

Return codes

Long. Return code choices (specify in a RETURN statement):

0 Continue processing

Usage

The Clicked event occurs when the user releases the mouse button.

If another control had focus, then a GetFocus and a Clicked event occur for the control the user clicks.

Examples

This code in an OLE control's Clicked event activates the object in the control:

```
integer li_success
li_success = This.Activate(InPlace!)
```

See also

GetFocus
RButtonDown

Close

The Close event has different arguments for different objects:

Object	See
Application	Syntax 1
OLE control	Syntax 2
Window	Syntax 3

Syntax 1 For the application object

Description

Occurs when the user closes the application.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Event ID

Event ID	Objects
None	Application

Arguments

None

Return codes

None (do not use a RETURN statement)

Usage

The Close event occurs when the last window (for MDI applications the MDI frame) is closed.

See also

Open
SystemError

Syntax 2 For OLE controls

Description

Occurs when the object in an OLE control has been activated offsite (the OLE server displays the object in the server's window) and that server is closed.

PocketBuilder	✗
PowerBuilder	✓

Syntax 3 For windows

Description

Occurs just before a window is removed from display.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Event ID

Event ID	Objects
pbm_close	Window

Arguments

None

Return codes

Long. Return code choices (specify in a RETURN statement):

0 Continue processing

Usage

When you call the Close function for the window, a CloseQuery event occurs before the Close event. In the CloseQuery event, you can specify a return code to prevent the Close event from occurring and the window from closing.

Do not trigger the Close event to close a window; call the Close function instead. Triggering the event simply runs the script and does not close the window.

See also

CloseQuery
Open

CloseQuery

Description

Occurs when a window is closed, before the Close event.

PocketBuilder	✗
PowerBuilder	✓

PocketBuilder applications
If your PocketBuilder application uses the Smart Minimize property, you can place the code that you put in the PowerBuilder CloseQuery event script in the Resize event script. Test that the sizetype argument of the Resize event is 1 before executing the code.

ColumnClick

Description

Occurs when the user clicks a column header.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✓

Event ID

Event ID	Objects
pbm_lvncolumnclick	ListView

Arguments

Argument	Description
column	The index of the clicked column

Return codes

Long. Return code choices (specify in a RETURN statement):

0 Continue processing

Usage

The ColumnClicked event is only available when the ListView displays in report view and the ButtonHeader property is set to true.

Examples

This example uses the ColumnClicked event to set up a instance variable for the column argument, retrieve column alignment information, and display it to the user:

```
string ls_label, ls_align
integer li_width
alignment la_align

ii_col = column
This.GetColumn(column, ls_label, la_align, &
   li_width)

CHOOSE CASE la_align
CASE Right!
   rb_right.Checked = TRUE
   ls_align = "Right!"

CASE Left!
   rb_left.Checked = TRUE
   ls_align = "Left!"
```

```
CASE Center!
   rb_center.Checked = TRUE
   ls_align = "Center!"

CASE Justify!
   rb_just.Checked = TRUE
   ls_align = "Justify!"
END CHOOSE

sle_info.Text = String(column) &
   + " " + ls_label &
   + " " + ls_align &
   + " " + String(li_width)
```

See also

Clicked

ConnectMobiLink

Description

Reserved for future use. Occurs when the MobiLink synchronization server connects to the consolidated database server.

Event ID

Event ID	Objects
None	MLSynchronization, MLSync

Arguments

None

Return codes

None

Constructor

Description

Occurs when the control or object is created, just before the Open event for the window that contains the control.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Event ID

Event ID	Objects
pbm_constructor	All objects

Arguments

None

Return codes

Long. Return code choices (specify in a RETURN statement):

0 Continue processing

Usage

You can write a script for a control's Constructor event to affect the control's properties before the window is displayed.

When a window or user object opens, a Constructor event for each control in the window or user object occurs. The order of controls in a window's Control property (which is an array) determines the order in which Constructor events are triggered. If one of the controls in the window is a user object, the Constructor events of all the controls in the user object occur before the Constructor event for the next control in the window.

When you call OpenUserObject to add a user object to a window dynamically, its Constructor event and the Constructor events for all of its controls occur.

When you use the CREATE statement to instantiate a class (nonvisual) user object, its Constructor event occurs.

When a class user object variable has an Autoinstantiate setting of true, its Constructor event occurs when the variable comes into scope. Therefore, the Constructor event occurs for:

- Global variables when the system starts up
- Shared variables when the object with the shared variables is loaded
- Instance variables when the object with the instance variables is created
- Local variables when the function that declares them begins executing

Examples

This example retrieves data for the DataWindow dw_1 before its window is displayed:

```
dw_1.SetTransObject(SQLCA)
dw_1.Retrieve( )
```

See also

Destructor
Open

DataChange

Description

Occurs when the server application notifies the control that data has changed.

PocketBuilder	✗
PowerBuilder	✓

Deactivate

Description

Occurs when the window becomes inactive.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Event ID

Event ID	Objects
pbm_deactivate	Window

Arguments

None

Return codes

Long. Return code choices (specify in a RETURN statement):

0 Continue processing

Usage

When a window is closed, a Deactivate event occurs.

See also

Activate
Show

DeleteAllItems

Description

Occurs when all the items in the ListView are deleted.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Event ID

Event ID	Objects

pbm_lvndeleteallitems	ListView

Arguments

None

Return codes

Long. Return code choices (specify in a RETURN statement):

0 Continue processing

Examples

This example uses the DeleteAllItems event to ensure that there is a default item in the ListView control:

```
This.AddItem("Default item", 1)
```

See also

DeleteItem
InsertItem

DeleteItem

The DeleteItem event has different arguments for different objects:

Object	See
ListView control	Syntax 1
TreeView control	Syntax 2

Syntax 1 For ListView controls

Description

Occurs when an item is deleted.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Event ID

Event ID	Objects
pbm_lvndeleteitem	ListView

Arguments

Argument	Description
index	Integer by value (the index of the deleted item)

Return codes

Long. Return code choices (specify in a RETURN statement):

0 Continue processing

Examples

This example for the DeleteItem event displays a message with the number of the deleted item:

```
MessageBox("Message", "Item " + String(index) &
   + " deleted.")
```

See also

DeleteAllItems
InsertItem

Syntax 2 For TreeView controls

Description

Occurs when an item is deleted.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Event ID

Event ID	Objects
pbm_tvndeleteitem	TreeView

Arguments

Argument	Description
handle	Long by value (the handle of the deleted item)

Return codes

Long. Return code choices (specify in a RETURN statement):

0 Continue processing

Examples

This example displays the name of the deleted item in a message:

```
TreeViewItem ll_tvi

This.GetItem(handle, ll_tvi)
MessageBox("Message", String(ll_tvi.Label) &
   + " has been deleted.")
```

Destructor

Description

Occurs when the user object or control is destroyed, immediately after the Close event of a window.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Event ID

Event ID	Objects
pbm_destructor	All objects

Arguments

None

Return codes

Long. Return code choices (specify in a RETURN statement):

0 Continue processing

Usage

When a window is closed, each control's Destructor event destroys the control and removes it from memory. After they have been destroyed, you can no longer refer to those controls in other scripts. If you do, a runtime error occurs.

See also

Constructor
Close

DisconnectMobiLink

Description

Reserved for future use. Occurs when the MobiLink synchronization server disconnects from the consolidated database server.

Event ID

Event ID	Objects
None	MLSynchronization, MLSync

Arguments

None

Return codes

None

DisplayMessage

Description

Reserved for future use. Occurs on display of an informational message from a MobiLink synchronization.

Event ID

Event ID	**Objects**
None	MLSynchronization, MLSync

Arguments

Argument	**Description**
infomsg	Read-only string containing the text of an informational message returned from the synchronization server.

Return codes

None

DoubleClicked

The DoubleClicked event has different arguments for different objects:

Object	**See**
ListBox, ListView, and Tab controls	Syntax 1
TreeView control	Syntax 2
Window	Syntax 3
Other controls	Syntax 4

For information about the DataWindow control’s DoubleClicked event, see the *DataWindow Reference* or the online Help.

Syntax 1 For ListBox, ListView, and Tab controls

Description

Occurs when the user double-clicks on the control.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✓

Event ID

Event ID	Objects
pbm_lbndblclk	ListBox
pbm_lvndoubleclicked	ListView
pbm_tcndoubleclicked	Tab

Arguments

Argument	Description
index	Integer by value. The index of the item the user double-clicked (for tabs, the index of the tab page).

Return codes

Long. Return code choices (specify in a RETURN statement):

0	Continue processing

Usage

In a ListBox, double-clicking on an item also triggers a SelectionChanged event.

Using the ItemActivate event for ListView controls
You can use the ItemActivate event (with the OneClickActivate property set to false) instead of the DoubleClicked event for ListView controls.

Examples

This example uses the DoubleClicked event to begin editing the double-clicked ListView item:

```
This.EditLabels = TRUE
```

See also

Clicked
ColumnClick
ItemActivate
ItemChanged
ItemChanging
RightClicked
RightDoubleClicked
SelectionChanged
SelectionChanging

Syntax 2 For TreeView controls

Description

Occurs when the user double-clicks on the control.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✓

Event ID

Event ID	Objects
pbm_tvndoubleclicked	TreeView

Arguments

Argument	Description
handle	Long by value (the handle of the item the user double-clicked)

Return codes

Long. Return code choices (specify in a RETURN statement):

0 Continue processing

Examples

This example turns on editing for the double-clicked TreeView item:

```
TreeViewItem ltvi_current
ltvi_current = tv_1.FindItem(CurrentTreeItem!, 0)
This.EditLabel(ltvi_current)
```

See also

Clicked
RightClicked
RightDoubleClicked
SelectionChanged
SelectionChanging

Syntax 3 For windows

Description

Occurs when the user double-clicks in an unoccupied area of the window (any area with no visible, enabled object).

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✓

Event ID

Event ID	Objects
pbm_lbuttondblclk	Window

Arguments

Argument	Description
flags	UnsignedLong by value (the modifier keys and mouse buttons that are pressed). Values are: • 1 — Left mouse button • 2 — Right mouse button • 4 — Shift key • 8 — Ctrl key • 16 — Middle mouse button In the Clicked event, the left mouse button is being released, so 1 is not summed in the value of *flags*. For an explanation of *flags*, see Syntax 2 of MouseMove on page 245.
xpos	Integer by value (the distance of the pointer from the left edge of the window's workspace in pixels).
ypos	Integer by value (the distance of the pointer from the top of the window's workspace in pixels).

Return codes

Long. Return code choices (specify in a RETURN statement):

0 Continue processing

Usage

The *xpos* and *ypos* arguments provide the same values the functions PointerX and PointerY return when you call them for the window.

See also

Clicked
MouseDown
MouseMove
MouseUp
RButtonDown

Syntax 4 For other controls

Description

Occurs when the user double-clicks on the control.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✓

Event ID

Event ID	Objects
pbm_bndoubleclicked	Graph, Picture, PictureHyperLink, StaticText, StaticHyperLink
pbm_prndoubleclicked	HProgressBar, VProgressBar

Arguments

None

Return codes

Long. Return code choices (specify in a RETURN statement):

0 Continue processing

Usage

The DoubleClicked event for DropDownListBoxes is only active when the Always Show List property is on.

See also

Clicked
RButtonDown

DragDrop

The DragDrop event has different arguments for different objects:

Object	See
ListBox, ListView, and Tab controls	Syntax 1
TreeView control	Syntax 2
Windows and other controls	Syntax 3

For information about the DataWindow control’s DragDrop event, see the *DataWindow Reference* or the online Help.

Syntax 1 For ListBox, ListView, and Tab controls

Description

Occurs when the user drags an object onto the control and releases the mouse button to drop the object.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✓

Event ID

Event ID	Objects
pbm_lbndragdrop	ListBox
pbm_lvndragdrop	ListView
pbm_tcndragdrop	Tab

Arguments

Argument	Description
source	DragObject by value (a reference to the control being dragged)
index	Integer by value (the index of the target ListView item)

Return codes

Long. Return code choices (specify in a RETURN statement):

0 Continue processing

Examples

For ListView controls, see the example for BeginDrag.

This example inserts the dragged ListView item:

```
This.AddItem(ilvi_dragged_object)
This.Arrange( )
```

See also

BeginDrag
BeginRightDrag
DragEnter
DragLeave
DragWithin

Syntax 2 For TreeView controls

Description

Occurs when the user drags an object onto the control and releases the mouse button to drop the object.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✓

Event ID

Event ID	Objects
pbm_tvndragdrop	TreeView

Arguments

Argument	Description
source	DragObject by value (a reference to the control being dragged)
handle	Long by value (the handle of the target TreeView item)

Return codes

Long. Return code choices (specify in a RETURN statement):

0 Continue processing

Examples

This example inserts the dragged object as a child of the TreeView item it is dropped upon:

```
TreeViewItem ltv_1
This.GetItem(handle, ltv_1)
This.SetDropHighlight(handle)
This.InsertItemFirst(handle, itvi_drag_object)
This.ExpandItem(handle)
This.SetRedraw(TRUE)
```

See also

DragEnter
DragLeave
DragWithin

Syntax 3 For windows and other controls

Description

Occurs when the user drags an object onto the control and releases the mouse button to drop the object.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✓

Event ID

Event ID	Objects
pbm_bndragdrop	CheckBox, CommandButton, Graph, Picture, PictureHyperLink, PictureButton, RadioButton
pbm_cbndragdrop	DropDownListBox
pbm_endragdrop	SingleLineEdit, EditMask, MultiLineEdit, StaticText, StaticHyperLink
pbm_prndragdrop	HProgressBar, VProgressBar
pbm_sbndragdrop	HScrollBar, HTrackBar, VScrollBar, VTrackBar
pbm_uondragdrop	UserObject
pbm_dragdrop	Window

Arguments

Argument	Description
source	DragObject by value (a reference to the control being dragged)

Return codes

Long. Return code choices (specify in a RETURN statement):

0 Continue processing

Usage

When a control's DragAuto property is true, a drag operation begins when the user presses a mouse button.

Examples

Example 1 In this example, the code in the DoubleClicked event for the DataWindow dw_orddetail starts a drag operation:

```
IF dw_orddetail.GetRow() > 0 THEN
   dw_orddetail.Drag(Begin!)
   This.DragIcon = "dragitem.ico"
END IF
```

Then, in the DragDrop event for a trashcan Picture control, this code deletes the row the user clicked and dragged from the DataWindow control:

```
long ll_currow
dwitemstatus ldwis_delrow
```

```
ll_currow = dw_orddetail.GetRow( )

// Save the row's status flag for later use
ldwis_delrow = dw_orddetail.GetItemStatus &
   (ll_currow, 0, Primary!)

// Now, delete the current row from dw_orddetail
dw_orddetail.DeleteRow(0)
```

Example 2 This example for a trashcan Picture control's DragDrop event checks whether the source of the drag operation is a DataWindow. If so, it asks the user whether to delete the current row in the source DataWindow:

```
DataWindow ldw_Source
Long ll_RowToDelete
Integer li_Choice

IF source.TypeOf() = DataWindow! THEN

   ldw_Source = source
   ll_RowToDelete = ldw_Source.GetRow()

   IF ll_RowToDelete > 0 THEN
      li_Choice = MessageBox("Delete", &
      "Delete this row?", Question!, YesNo!, 2)
      IF li_Choice = 1 THEN
      ldw_Source.DeleteRow(ll_RowToDelete)
      END IF
   ELSE
      Beep(1)
   END IF

ELSE
   Beep(1)
END IF
```

See also

DragEnter
DragLeave
DragWithin

DragEnter

Description

Occurs when the user is dragging an object and enters the control.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✓

Event ID

Event ID	Objects
pbm_bndragenter	CheckBox, CommandButton, Graph, Picture, PictureHyperlink, PictureButton, RadioButton
pbm_cbndragenter	DropDownListBox
pbm_dwndragenter	DataWindow
pbm_endragenter	SingleLineEdit, EditMask, MultiLineEdit, StaticText, StaticHyperLink
pbm_lbndragenter	ListBox
pbm_lvndragenter	ListView
pbm_prndragenter	HProgressBar, VProgressBar
pbm_sbndragenter	HScrollBar, HTrackBar, VScrollBar, VTrackBar
pbm_tcndragenter	Tab
pbm_tvndragenter	TreeView
pbm_uondragenter	UserObject
pbm_dragenter	Window

Arguments

Argument	Description
source	DragObject by value (a reference to the control being dragged)

Return codes

Long. Return code choices (specify in a RETURN statement):

0 Continue processing

Examples

This example for a Picture control's DragDrop event adds a border to itself when another Picture control (the source) is dragged within its boundaries:

```
IF source.TypeOf() = Picture! THEN
   This.Border = TRUE
END IF
```

See also

DragDrop
DragLeave
DragWithin

DragLeave

Description

Occurs when the user is dragging an object and leaves the control.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✓

Event ID

Event ID	Objects
pbm_bndragleave	CheckBox, CommandButton, Graph, Picture, PictureHyperLink, PictureButton, RadioButton
pbm_cbndragleave	DropDownListBox
pbm_dwndragleave	DataWindow
pbm_endragleave	SingleLineEdit, EditMask, MultiLineEdit, StaticText, StaticHyperLink
pbm_lbndragleave	ListBox
pbm_lvndragleave	ListView
pbm_prndragleave	HProgressBar, VProgressBar
pbm_sbndragleave	HScrollBar, HTrackBar, VScrollBar, VTrackBar
pbm_tcndragleave	Tab
pbm_tvndragleave	TreeView
pbm_uondragleave	UserObject
pbm_dragleave	Window

Arguments

Argument	Description
source	DragObject by value (a reference to the control being dragged)

Return codes

Long. Return code choices (specify in a RETURN statement):

0 Continue processing

Examples

This example checks the name of the control being dragged, and if it is, cb_1 it cancels the drag operation:

```
IF ClassName(source) = "cb_1" THEN
   cb_1.Drag(Cancel!)
END If
```

This example for a Picture control's DragDrop event removes its own border when another Picture control (the source) is dragged beyond its boundaries:

```
IF source.TypeOf() = Picture! THEN
   This.Border = TRUE
END IF
```

See also

DragDrop
DragEnter
DragWithin

DragWithin

The DragWithin event has different arguments for different objects:

Object	See
ListBox, ListView, and Tab controls	Syntax 1
TreeView control	Syntax 2
Windows and other controls	Syntax 3

For information about the DataWindow control's DragWithin event, see the *DataWindow Reference* or the online Help.

Syntax 1 For ListBox, ListView, and Tab controls

Description

Occurs when the user is dragging an object within the control.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✓

Event ID

Event ID	Objects
pbm_lbndragwithin	ListBox
pbm_lvndragwithin	ListView
pbm_tcndragwithin	Tab

Arguments

Argument	Description
source	DragObject by value (a reference to the control being dragged)
index	Integer by value (a reference to the ListView item under the pointer in the ListView control)

Return codes

Long. Return code choices (specify in a RETURN statement):

0 Continue processing

Examples

This example changes the background color of the ListView when a DragObject enters its border:

```
This.BackColor = RGB(128, 0, 128)
```

See also

DragDrop
DragEnter
DragLeave

Syntax 2 For TreeView controls

Description

Occurs when the user is dragging an object within the control.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✓

Event ID

Event ID	Objects
pbm_tvndragwithin	TreeView

Arguments

Argument	Description
source	DragObject by value (a reference to the control being dragged)
handle	Long (a reference to the ListView item under the pointer in the TreeView control)

Return codes

Long. Return code choices (specify in a RETURN statement):

0 Continue processing

Examples

This example changes the background color of the TreeView when a DragObject enters its border:

```
This.BackColor = RGB(128, 0, 128)
```

See also

DragDrop
DragEnter
DragLeave

Syntax 3 For windows and other controls

Description

Occurs when the user is dragging an object within the control.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✓

Event ID

Event ID	Objects
pbm_bndragwithin	CheckBox, CommandButton, Graph, Picture, PictureHyperLink, PictureButton, RadioButton
pbm_cbndragwithin	DropDownListBox
pbm_endragwithin	SingleLineEdit, EditMask, MultiLineEdit, StaticText, StaticHyperLink
pbm_prndragwithin	HProgressBar, VProgressBar
pbm_sbndragwithin	HScrollBar, HTrackBar, VScrollBar, VTrackBar
pbm_uondragwithin	UserObject
pbm_dragwithin	Window

Arguments

Argument	Description
source	DragObject by value (a reference to the control being dragged)

Return codes

Long. Return code choices (specify in a RETURN statement):

0 Continue processing

See also

DragDrop
DragEnter
DragLeave

EndDownload

Description Reserved for future use. Occurs at the end of a download procedure

Event ID

Event ID	Objects
None	MLSynchronization, MLSync

Arguments

Argument	Description
upsertrows	Long identifying the inserted and updated rows.
deleterows	Long identifying the deleted rows.

Return codes None

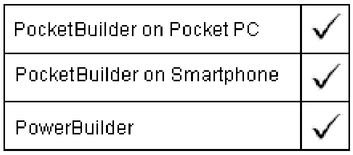
EndLabelEdit

The EndLabelEdit event has different arguments for different objects:

Object	See
ListView control	Syntax 1
TreeView control	Syntax 2

Syntax 1 For ListView controls

Description Occurs when the user finishes editing an item's label.



PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Event ID

Event ID	Objects
pbm_lvnendlabeledit	ListView

Arguments

Argument	Description
index	Integer. The index of the ListView item for which you have edited the label.

Argument	Description
newlabel	The string that represents the new label for the ListView item.

Return codes

Long. Return code choices (specify in a RETURN statement):

0 Allow the new text to become the item's label.
1 Prevent the new text from becoming the item's label.

Usage

The user triggers this event by pressing Enter or Tab after editing the text.

Examples

This example displays the old label and the new label in a SingleLineEdit:

```
ListViewItem lvi
sle_info.text = "Finished editing " &
   + String(lvi.label) &
   +". Item changed to "+ String(newlabel)
```

See also

BeginLabelEdit

Syntax 2 For TreeView controls

Description

Occurs when the user finishes editing an item's label.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Event ID

Event ID	Objects
pbm_tvnendlabeledit	TreeView

Arguments

Argument	Description
handle	Integer. The index of the TreeView item for which you have edited the label.
newtext	The string that represents the new label for the TreeView item.

Return codes

Long. Return code choices (specify in a RETURN statement):

0 Allow the new text to become the item's label
1 Prevent the new text from becoming the item's label

Usage

The user triggers this event by pressing Enter or Tab after editing the text.

Examples

This example displays the old label and the new label in a SingleLineEdit:

```
TreeViewItem tvi

This.GetItem(handle, tvi)
sle_info.Text = "Finished editing " &
   + String(tvi.Label) &
   + ". Item changed to " &
   + String(newtext)
```

See also

BeginLabelEdit

EndLogScan

Description

Reserved for future use. Occurs after the scan of the transaction log completes for upload.

Event ID

Event ID	**Objects**
None	MLSync

Arguments

None

Return codes

None

EndSync

Description

Reserved for future use. Occurs at the end of synchronization.

Event ID

Event ID	**Objects**
None	MLSynchronization, MLSync

Arguments

Argument	**Description**
rc	Long datatype value that indicates whether a synchronization error occurred.
restart	Boolean value passed by reference that, if true, causes dbmlsync to restart the syncrhonization.

Return codes

None

EndUpload

Description

Reserved for future use. Occurs after transmission of the upload to the synchronization server.

Event ID

Event ID	Objects
None	MLSynchronization, MLSync

Arguments

None

Return codes

None

Error

Description

Occurs when an error is found in a data or property expression for an external object or a DataWindow object. Also occurs when a communications error is found in a client connecting to EAServer.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Improved error-handling capability
The Error event is maintained for backward compatibility only. If you do not script the Error event or change its action argument, information from this event is passed to RuntimeError objects, such as DWRuntimeError. You can handle these errors in a try-catch block.

Event ID

Event ID	Objects
None	Connection, DataWindow, DataStore

Arguments

Argument	Description
errornumber	Unsigned integer by value (PocketBuilder's error number)
errortext	String, read-only (PocketBuilder's error message)
errorwindowmenu	String, read-only (the name of the window or menu that is the parent of the object whose script caused the error)
errorobject	String, read-only (the name of the object whose script caused the error)
errorscript	String, read-only (the full text of the script in which the error occurred)
errorline	Unsigned integer by value (the line in the script where the error occurred)
action	ExceptionAction by reference. A value you specify to control the application's course of action as a result of the error. Values are: • ExceptionFail! — fail as if this script were not implemented. The error condition triggers any active event handlers, or if none, the SystemError event. • ExceptionIgnore! — ignore this error and return as if no error occurred (use this option with caution because the conditions that caused the error can cause another error). • ExceptionRetry! — execute the function or evaluate the expression again in case the OLE server was not ready. This option is not valid for DataWindows. • ExceptionSubstituteReturnValue! — use the value specified in the *returnvalue* argument instead of the value returned by the OLE server or DataWindow, and cancel the error condition.
returnvalue	Any by reference (a value whose datatype matches the expected value that the OLE server or DataWindow would have returned). This value is used when the value of *action* is ExceptionSubstituteReturnValue!.

Return codes

None. Do not use a RETURN statement.

Examples

This example displays information about the error that occurred and allows the script to continue:

```
MessageBox("Error Number " + string(errornumber)&
   + " Occurred", "Errortext: " + String(errortext))
action = ExceptionIgnore!
```

See also

DBError in the *DataWindow Reference* or the online Help
ExternalException
SystemError

ErrorMessage

Description

Reserved for future use. Occurs on display of an error message from a MobiLink synchronization.

Event ID

Event ID	Objects
None	MLSynchronization, MLSync

Arguments

Argument	Description
errmsg	Read-only string containing the text of the error message returned from the synchronization server.

Return codes

None

ExternalException

Description

Occurs when an OLE automation command caused an exception on the OLE server.

PocketBuilder	✗
PowerBuilder	✓

Event ID

Event ID	Objects
None	OLE, OLEObject, OLETxnObject

Return codes

None. (Do not use a RETURN statement.)

FileExists

Description

Occurs when a file is saved in the RichTextEdit control and the file already exists.

PocketBuilder	✗
PowerBuilder	✓

Event ID

Event ID	**Objects**
pbm_renfileexists	RichTextEdit

Arguments

Argument	**Description**
filename	The name of the file

Return codes

Long. Return code choices (specified in a RETURN statement):

0	Continue processing
1	Saving of document is canceled

FileMessage

Description

Reserved for future use. Occurs on display of a detailed information message from a MobiLink synchronization.

Event ID

Event ID	**Objects**
None	MLSynchronization, MLSync

Arguments

Argument	**Description**
filemsg	Read-only string containing the text of the message returned from the synchronization server.

Return codes

None

GetFocus

Description

Occurs just before the control receives focus (before it is selected and becomes active).

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

GetFocus applies to all controls

Event ID

Event ID	Objects
pbm_bnsetfocus	CheckBox, CommandButton, Graph, Picture, PictureHyperLink, PictureButton, RadioButton
pbm_cbnsetfocus	DropDownListBox
pbm_dwnsetfocus	DataWindow
pbm_ensetfocus	SingleLineEdit, EditMask, MultiLineEdit, StaticText, StaticHyperLink
pbm_lbnsetfocus	ListBox
pbm_lvnsetfocus	ListView
pbm_prnsetfocus	HProgressBar, VProgressBar
pbm_sbnsetfocus	HScrollBar, HTrackBar, VScrollBar, VTrackBar
pbm_tcnsetfocus	Tab
pbm_tvnsetfocus	TreeView

Arguments

None

Return codes

Long. Return code choices (specified in a RETURN statement):

0 Continue processing

Examples

Example 1 This example in a SingleLineEdit control's GetFocus event selects the text in the control when the user tabs to it:

```
This.SelectText(1, Len(This.Text))
```

Example 2 In Example 1, when the user clicks the SingleLineEdit rather than tabbing to it, the control gets focus and the text is highlighted, but then the click deselects the text. If you define a user event that selects the text and then post that event in the GetFocus event, the highlighting works when the user both tabs and clicks. This code is in the GetFocus event:

```
This. EVENT POST ue_select( )
```

This code is in the ue_select user event:

```
This.SelectText(1, Len(This.Text))
```

See also

Clicked
LoseFocus

Help

Description

Occurs when the user drags the question-mark button from the title bar to a menu item or a control and then clicks, or when the user clicks in a control (giving it focus) and then presses the F1 key.

PocketBuilder	✗
PowerBuilder	✓

Event ID

Event ID	Objects
pbm_help	Window, Menu, DragObject

Arguments

Argument	Description
xpos	Integer by value (the distance of the Help message from the left edge of the screen, in PowerBuilder units)
ypos	Integer by value (the distance of the Help message from the top of the screen, in PowerBuilder units)

Return codes

Long. Return code choices (specified in a RETURN statement):

0	Continue processing

Hide

Description

Occurs just before the window is hidden.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Event ID

Event ID	Objects
pbm_hidewindow	Window

Arguments None

Return codes Long. Return code choices (specified in a RETURN statement):

0 Continue processing

Usage A Hide event can occur when a sheet in an MDI frame is closed. It does not occur when closing a main, response, or pop-up window.

See also Close
Show

HotLinkAlarm

Description Occurs after a Dynamic Data Exchange (DDE) server application has sent new (changed) data and the client DDE application has received it.

PocketBuilder	✗
PowerBuilder	✓

Event ID

Event ID	Objects
pbm_ddedata	Window

Arguments None

Return codes Long. Return code choices (specify in a RETURN statement):

0 Continue processing

Idle

Description

Occurs when the Idle function has been called in an application object script and the specified number of seconds have elapsed with no mouse or keyboard activity.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Event ID

Event ID	Objects
None	Application

Arguments

None

Return codes

None. (Do not use a RETURN statement.)

Examples

This statement in an application script causes the Idle event to be triggered after 300 seconds of inactivity:

```
Idle(300)
```

In the Idle event itself, this statement closes the application:

```
HALT CLOSE
```

IncomingMessage

Description

Occurs when an incoming SMS message is received.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✗

Event ID

Event ID	Objects
None	SMSSession

Arguments

Argument	Description
SMSAddress	Object of type SMSAddress that contains the address and address type from which the incoming message originated
SMSMessage	Object of type SMSMessage that includes the message ID and content of the incoming SMS message

Return codes

Boolean. A return value of true is interpreted as a request to deletes the incoming message. A return value of false places the message in the SMS inbox. If no return value is specified, the default is false.

Usage

All instantiated SMSSession objects listening for messages are notified of an incoming SMS message. PocketBuilder can prevent the message from displaying in the SMS inbox if you set the IncomingMessage event to return true. If any SMSSession object requests deletion of the message by returning true for the IncomingMessage event, PocketBuilder will attempt to delete the message, but only after all SMSSession objects have processed the message.

Deletion of messages is dependent on registry setting
PocketBuilder applications can receive notification of an SMS message only after you register the shim DLL, *PKSMS25.DLL*, on your device. The ReadOnly registry attribute for the DLL must be set to 0 before PocketBuilder can delete an SMS message For information on registering the DLL and the ReadOnly registry attribute, see the chapter on Working with Native Objects and Controls in the *Users Guide*.

The IncomingMessage event is synchronous with the operating system processing SMS messages. Therefore you should not include code that prompts for user input, or perform any lengthy operation in the script for this event.

Examples

The following code in the IncomingMessage event determines whether an incoming SMS message is placed in the SMS inbox or is deleted based on whether or not the message contains the text “top secret”:

```
// If the message contains "top secret" it will deleted.
// Otherwise, it is placed in the inbox.

if POS(SMSMsg.text, "top secret") <> 0 then
    // delete this e-mail and notify user of deletion
   //  in a MultiLineEdit control
    mle_status.text += "~r~nWill be deleted.~r~n"
    return TRUE
end if
```

```
// allow to go into the inbox
return FALSE
```

InputFieldSelected

Description

In a RichTextEdit control, occurs when the user has double-clicked or pressed Enter in an input field, allowing the user to edit the data in the field.

PocketBuilder	✗
PowerBuilder	✓

Event ID

Event ID	Objects
pbm_reninputfieldselected	RichTextEdit

Arguments

Argument	Description
fieldname	String by value (the name of the input field that was selected)

Return codes

Long. Return code choices (specify in a RETURN statement):

0 Continue processing

InsertItem

Description

Occurs when an item is inserted in the ListView.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Event ID

Event ID	Objects
pbm_lvninsertitem	ListView

Arguments

Argument	Description
index	An integer that represents the index of the item being inserted into the ListView

Return codes

Long. Return code choices (specified in a RETURN statement):

0 Continue processing

Examples

This example displays the label and index of the inserted item:

```
ListViewItem lvi
This.GetItem(index, lvi)
sle_info.Text = "Inserted "+ String(lvi.Label) &
   + " into position " &
   + String(index)
```

See also

DeleteItem

ItemActivate

Description

Occurs when a ListView item is clicked or double-clicked. The actual firing mechanism depends on the OneClickActivate and TwoClickActivate property settings.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Event ID

Event ID	Objects
pbm_lvnitemactivate	ListView

Arguments

Argument	Description
Index	An integer that represents the index of the item being inserted into the ListView

Return codes

Long. Return code choices (specify in a RETURN statement):

0 Continue processing

Usage

Use the ItemActivate event instead of the Clicked or DoubleClicked event in new applications.

The following ListView property settings determine which user action fires the event:

OneClickActivate	TwoClickActivate	Firing mechanism
True	True	Single click
True	False	Single click
False	True	Single click on selected item or double-click on nonselected item
False	False	Double-click

Examples

This code changes a ListView item text label to uppercase lettering. The change is made in the second column of the item the user clicks or double-clicks, depending on the ListView property settings:

```
listviewitem llvi_current

This.GetItem(index, 2, llvi_current)
llvi_current.Label = Upper(llvi_current.Label)
This.SetItem(index, 2, llvi_current)
RETURN 0
```

See also

ItemChanged
ItemChanging

ItemChanged

Description

Occurs when an ListView item has changed.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Event ID

Event ID	Objects
pbm_lvnitemchanged	ListView

Arguments

Argument	Description
index	The index of the item that is changing
focuschanged	Boolean (specifies if focus has changed for the item)

Argument	Description
hasfocus	Boolean (specifies whether the item has focus)
selectionchange	Boolean (specifies whether the selection has changed for the item)
selected	Boolean (specifies whether the item is selected)
otherchange	Boolean (specifies if anything other than focus or selection has changed for the item)

Return codes

Long. Return code choices (specify in a RETURN statement):

0 Continue processing

Examples

This example checks whether the event is occurring because focus has changed to the item:

```
ListViewItem l_lvi

lv_list.GetItem(index, l_lvi)
IF focuschange and hasfocus THEN
   sle1.Text = String(lvi.label) +" has focus."
END IF
```

See also

ItemChanged in the *DataWindow Reference* or the online Help
ItemChanging

ItemChanging

Description

Occurs just before a ListView changes.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Event ID

Event ID	Objects
pbm_lvnitemchanging	ListView

Arguments

Argument	Description
index	The index of the item that has changed
focuschange	Boolean (specifies if focus is changing for the item)

Argument	Description
hasfocus	Boolean (specifies whether the item has focus)
selectionchange	Boolean (specifies whether the selection is changing for the item)
selected	Boolean (specifies whether the item is selected)
otherchange	Boolean (specifies if anything other than focus or selection has changed for the item)

Return codes

Long. Return code choices (specify in a RETURN statement):

0 Continue processing

See also

ItemChanged

ItemCollapsed

Description

Occurs when a TreeView item has collapsed.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Event ID

Event ID	Objects
pbm_tvnitemcollapsed	TreeView

Arguments

Argument	Description
handle	Long by reference (the handle of the collapsed TreeViewItem)

Return codes

Long. Return code choices (specified in a RETURN statement):

0 Continue processing

Examples

This example changes the picture for the collapsed item:

```
TreeViewItem l_tvi
integer li_level

This.GetItem(handle, l_tvi)

CHOOSE CASE l_tvi.Level
```

```
          CASE 1
             l_tvi.PictureIndex = 1
             l_tvi.SelectedPictureIndex = 1
          CASE 2
             l_tvi.PictureIndex = 2
             l_tvi.SelectedPictureIndex = 2
          CASE 3
             l_tvi.PictureIndex = 3
             l_tvi.SelectedPictureIndex = 3
          CASE 4
             l_tvi.PictureIndex = 4
             l_tvi.SelectedPictureIndex = 4
       END CHOOSE
       This.SetItem(handle, l_tvi)
```

See also ItemCollapsing

ItemCollapsing

Description Occurs when a TreeView item is collapsing.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Event ID

Event ID	Objects
pbm_tvnitemcollapsing	TreeView

Arguments

Argument	Description
handle	Long by reference (the handle of the collapsing item)

Return codes Long. Return code choices (specify in a RETURN statement):

0 Continue processing

Usage The ItemCollapsing event occurs before the ItemCollapsed event.

Examples This example changes the picture for the collapsing item:

```
TreeViewItem l_tvi
integer li_level
```

```
This.GetItem(handle, l_vti)

CHOOSE CASE l_tvi.level
   CASE 1
      l_tvi.PictureIndex = 1
      l_tvi.SelectedPictureIndex = 1
   CASE 2
      l_tvi.PictureIndex = 2
      l_tvi.SelectedPictureIndex = 2
   CASE 3
      l_tvi.PictureIndex = 3
      l_tvi.SelectedPictureIndex = 3
   CASE 4
      l_tvi.PictureIndex = 4
      l_tvi.SelectedPictureIndex = 4
END CHOOSE

This.SetItem(handle, l_tvi)
```

See also

ItemCollapsed

ItemExpanded

Description

Occurs when a TreeView item has expanded.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Event ID

Event ID	Objects
pbm_tvnitemexpanded	TreeView

Arguments

Argument	Description
handle	Long by reference (the handle of the expanded item)

Return codes

Long. Return code choices (specify in a RETURN statement):

0 Continue processing

Usage

The ItemExpanded event occurs after the ItemExpanding event.

Examples

This example sets the picture and selected picture for the expanded item:

```
TreeViewItem l_tvi
integer li_level

This.GetItem(handle, l_tvi)

CHOOSE CASE l_tvi.Level
   CASE 1
      l_tvi.PictureIndex = 5
      l_tvi.SelectedPictureIndex = 1
   CASE 2
      l_tvi.PictureIndex = 5
      l_tvi.SelectedPictureIndex = 2
   CASE 3
      l_tvi.PictureIndex = 5
      l_tvi.SelectedPictureIndex = 3
   CASE 4
      l_tvi.PictureIndex = 4
      l_tvi.SelectedPictureIndex = 5
END CHOOSE
This.SetItem(handle, l_tvi)
```

See also

ItemExpanding

ItemExpanding

Description

Occurs *while* a TreeView item is expanding.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Event ID

Event ID	Objects
pbm_tvnitemexpanding	TreeView

Arguments

Argument	Description
handle	Long by reference (the handle of the expanding TreeView item)

Return codes

Long. Return code choices (specify in a RETURN statement):

0	Continue processing
1	Prevents the TreeView from expanding

Usage

The ItemExpanding event occurs *before* the ItemExpanded event.

Examples

This example sets the picture and selected picture for the expanding item:

```
TreeViewItem l_tvi
integer li_level

This.GetItem(handle, l_tvi)

CHOOSE CASE l_tvi.Level
   CASE 1
      l_tvi.PictureIndex = 5
      l_tvi.SelectedPictureIndex = 1
   CASE 2
      l_tvi.PictureIndex = 5
      l_tvi.SelectedPictureIndex = 2
   CASE 3
      l_tvi.PictureIndex = 5
      l_tvi.SelectedPictureIndex = 3
   CASE 4
      l_tvi.PictureIndex = 4
      l_tvi.SelectedPictureIndex = 5
END CHOOSE

This.SetItem(handle, l_tvi)
```

See also

ItemExpanded

ItemPopulate

Description

Occurs when a TreeView item is being populated with children.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Event ID

Event ID	**Objects**
pbm_tvnitempopulate	TreeView

Arguments

Argument	Description
handle	Long by reference (the handle of the TreeView item being populated)

Return codes

Long. Return code choices (specified in a RETURN statement):

0 Continue processing

Examples

This example displays the name of the TreeView item you are populating in a SingleLineEdit:

```
TreeViewItem tvi

This.GetItem(handle, tvi)
sle_get.Text = "Populating TreeView item " &
   + String(tvi.Label) + " with children"
```

See also

ItemExpanding

Key

Description

Occurs when the user presses a key.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Event ID

Event ID	Objects
pbm_lvnkeydown	ListView
pbm_renkey	RichTextEdit
pbm_tcnkeydown	Tab
pbm_tvnkeydown	TreeView
pbm_keydown	Window

Arguments

Argument	Description
key	KeyCode by value. A value of the KeyCode enumerated datatype indicating the key that was pressed (for example, KeyA! or KeyF1!).

Argument	Description
keyflags	UnsignedLong by value (the modifier keys that were pressed with the key). Values are: 1 Shift key 2 Ctrl key 3 Shift and Ctrl keys

Return codes

Long. Return code choices (specify in a RETURN statement):

0	Continue processing
1	Do not process the key (RichTextEdit controls only)

Usage

Some controls capture keystrokes so that the window is prevented from getting a Key event. These include ListView, TreeView, Tab, RichTextEdit, and the DataWindow edit control. When these controls have focus you can respond to keystrokes by writing a script for an event for the control. If there is no predefined event for keystrokes, you can define a user event and associate it with a pbm code.

If the user presses a modifier key and holds it down while pressing another key, the Key event occurs twice: once when the modifier key is pressed and again when the second key is pressed. If the user releases the modifier key before pressing the second key, the value of *keyflags* will change in the second occurrence.

When the user releases a key, the Key event does not occur. Therefore, if the user releases a modifier key, you do not know the current state of the modifier keys until another key is pressed.

Examples

This example causes a beep when the user presses F1 or F2, as long as Shift and Ctrl are not pressed:

```
IF keyflags = 0 THEN
   IF key = KeyF1! THEN
      Beep(1)
   ELSEIF key = KeyF2! THEN
      Beep(20)
   END IF
END IF
```

This line displays the value of *keyflags* when a key is pressed.

```
st_1.Text = String(keyflags)
```

See also

SystemKey

LineDown

Description

Occurs when the user clicks the down arrow of the vertical scroll bar.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Event ID

Event ID	Objects
pbm_sbnlinedown	VScrollBar, VTrackBar

Arguments

None

Return codes

Long. Return code choices (specify in a RETURN statement):

0 Continue processing

Usage

When the user clicks in a vertical scroll bar, nothing happens unless you have scripts that change the scroll bar's Position property. For the scroll bar arrows, use the LineUp and LineDown events; for clicks in the scroll bar background above and below the thumb, use the PageUp and PageDown event; for dragging the thumb itself, use the Moved event.

Examples

This code in the LineDown event causes the thumb to move down when the user clicks on the down arrow of the vertical scroll bar and displays the resulting position in the StaticText control st_1:

```
IF This.Position > This.MaxPosition - 1 THEN
   This.Position = MaxPosition
ELSE
   This.Position = This.Position + 1
END IF

st_1.Text = "LineDown " + String(This.Position)
```

See also

LineLeft
LineRight
LineUp
PageDown

LineLeft

Description

Occurs when the user clicks in the left arrow of the horizontal scroll bar.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Event ID

Event ID	Objects
pbm_sbnlineup	HScrollBar, HTrackBar

Arguments

None

Return codes

Long. Return code choices (specify in a RETURN statement):

0 Continue processing

Usage

When the user clicks in a horizontal scroll bar, nothing happens unless you have scripts that change the scroll bar's Position property. For the scroll bar arrows, use the LineLeft and LineRight events; for clicks in the scroll bar background above and below the thumb, use the PageLeft and Right events; for dragging the thumb itself, use the Moved event.

Examples

This code in the LineLeft event causes the thumb to move left when the user clicks on the left arrow of the horizontal scroll bar and displays the resulting position in the StaticText control st_1:

```
IF This.Position < This.MinPosition + 1 THEN
   This.Position = MinPosition
ELSE
   This.Position = This.Position - 1
END IF

st_1.Text = "LineLeft " + String(This.Position)
```

See also

LineDown
LineRight
LineUp
PageLeft

LineRight

Description

Occurs when right arrow of the horizontal scroll bar is clicked.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Event ID

Event ID	Objects
pbm_sbnlinedown	HScrollBar, HTrackBar

Arguments

None

Return codes

Long. Return code choices (specify in a RETURN statement):

0 Continue processing

Usage

When the user clicks in a horizontal scroll bar, nothing happens unless you have scripts that change the scroll bar's Position property. For the scroll bar arrows, use the LineLeft and LineRight events; for clicks in the scroll bar background above and below the thumb, use the PageLeft and PageRight events; for dragging the thumb itself, use the Moved event.

Examples

This code in the LineRight event causes the thumb to move right when the user clicks on the right arrow of the horizontal scroll bar and displays the resulting position in the StaticText control st_1:

```
IF This.Position > This.MaxPosition - 1 THEN
   This.Position = MaxPosition
ELSE
   This.Position = This.Position + 1
END IF

st_1.Text = "LineRight " + String(This.Position)
```

See also

LineDown
LineLeft
LineUp
PageRight

LineUp

Description

Occurs when the up arrow of the vertical scroll bar is clicked.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Event ID

Event ID	Objects
pbm_sbnlineup	VScrollBar, VTrackBar

Arguments

None

Return codes

Long. Return code choices (specify in a RETURN statement):

0 Continue processing

Usage

When the user clicks in a vertical scroll bar, nothing happens unless you have scripts that change the scroll bar's Position property. For the scroll bar arrows, use the LineUp and LineDown events; for clicks in the scroll bar background above and below the thumb, use the PageUp and PageDown events; for dragging the thumb itself, use the Moved event.

Examples

This code in the LineUp event causes the thumb to move up when the user clicks on the up arrow of the vertical scroll bar and displays the resulting position in the StaticText control st_1:

```
IF This.Position < This.MinPosition + 1 THEN
   This.Position = MinPosition
ELSE
   This.Position = This.Position - 1
END IF

st_1.Text = "LineUp " + String(This.Position)
```

See also

LineDown
LineLeft
LineRight
PageUp

LoseFocus

Description

Occurs just before a control receives focus (before it becomes selected and active).

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Event ID

Event ID	**Description**
pbm_*controltype*killfocus	UserObject (standard visual user objects only)
pbm_bnkillfocus	CheckBox, CommandButton, Graph, OLE, Picture, PictureHyperLink, PictureButton, RadioButton, StaticText, StaticHyperLink
pbm_cbnkillfocus	DropDownListBox, DropDownPictureListBox
pbm_dwnkillfocus	DataWindow
pbm_enkillfocus	SingleLineEdit, EditMask, MultiLineEdit
pbm_lbnkillfocus	ListBox, PictureListBox
pbm_lvnkillfocus	ListView
pbm_prnkillfocus	HProgressBar, VProgressBar
pbm_renkillfocus	RichTextEdit
pbm_sbnkillfocus	HScrollBar, HTrackBar, VScrollBar, VTrackBar
pbm_tcnkillfocus	Tab
pbm_tvnkillfocus	TreeView

Arguments

None

Return codes

Long. Return code choices (specify in a RETURN statement):

0	Continue processing

Usage

Write a script for a control's LoseFocus event if you want some processing to occur when the user changes focus to another control.

For controls that contain editable text, losing focus can also cause a Modified event to occur.

Because the MessageBox function grabs focus, you should not use it when focus is changing, such as in a LoseFocus event. Instead, you might display a message in the window's title or a MultiLineEdit.

Examples

Example 1 In this script for the LoseFocus event of a SingleLineEdit sle_town, the user is reminded to enter information if the text box is left empty:

```
IF sle_town.Text = "" THEN
   st_status.Text = "You have not specified a town."
END IF
```

Example 2 Statements in the LoseFocus event for a DataWindow control dw_emp can trigger a user event whose script validates the last item the user entered.

This statement triggers the user event ue_accept:

```
dw_emp.EVENT ue_accept( )
```

This statement in ue_accept calls the AcceptText function:

```
dw_emp.AcceptText( )
```

This script for the LoseFocus event of a RichTextEdit control performs processing when the control actually loses focus:

```
GraphicObject l_control

// Check whether the RichTextEdit still has focus
l_control = GetFocus()
IF TypeOf(l_control) = RichTextEdit! THEN RETURN 0

// Perform processing only if RichTextEdit lost focus
...
```

This script gets the name of the control instead:

```
GraphicObject l_control
string ls_name
l_control = GetFocus()
ls_name = l_control.Classname( )
```

See also

GetFocus

Modified

Description

Occurs when the contents in the control has changed.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Event ID

Event ID	Objects
pbm_cbnmodified	DropDownListBox, DropDownPictureListBox
pbm_enmodified	SingleLineEdit, EditMask, MultiLineEdit
pbm_renmodified	RichTextEdit

Arguments

None

Return codes

Long. Return code choices (specify in a RETURN statement):

0 Continue processing

Usage

For plain text controls, the Modified event occurs when the user indicates being finished by pressing Enter or tabbing away from the control.

For RichText Edit controls, the value of the Modified property controls the Modified event. If the property is false, the event occurs when the first change occurs to the contents of the control. The change also causes the property to be set to true, which suppresses the Modified event. You can restart checking for changes by setting the property back to false.

Resetting the Modified property is useful when you insert a document in the control, which triggers the event and sets the property (it is reporting the change to the control's contents). To find out when the user begins making changes to the content, set the Modified property back to false in the script that opens the document. When the user begins editing, the property will be reset to true and the event will occur again.

A Modified event can be followed by a LoseFocus event.

Examples

In this example, code in the Modified event performs validation on the text the user entered in a SingleLineEdit control sle_color. If the user did not enter RED, WHITE, or BLUE, a message box indicates what is valid input; for valid input, the color of the text changes:

```
string ls_color

This.BackColor = RGB(150,150,150)
```

```
ls_color = Upper(This.Text)
CHOOSE CASE ls_color
   CASE "RED"
      This.TextColor = RGB(255,0,0)
   CASE "BLUE"
      This.TextColor = RGB(0,0,255)
   CASE "WHITE"
      This.TextColor = RGB(255,255,255)
   CASE ELSE
      This.Text = ""
      MessageBox("Invalid input", &
      "Enter RED, WHITE, or BLUE.")
END CHOOSE
```

This is not a realistic example: user input of three specific choices is more suited to a list box; in a real situation, the allowed input might be more general.

See also LoseFocus

MouseDown

The MouseDown event has different arguments for different objects:

Object	See
RichTextEdit control	Syntax 1
Window	Syntax 2

Syntax 1 For RichTextEdit controls

Description

Occurs when the user presses the left mouse button on the RichTextEdit control.

PocketBuilder	✗
PowerBuilder	✓

Event ID

Event ID	Objects
pbm_renlbuttondown	RichTextEdit

Arguments

None

Return codes

Long. Return code choices (specify in a RETURN statement):

0 Continue processing

Syntax 2 For windows

Description

Occurs when the user presses the left mouse button in an unoccupied area of the window (any area with no visible, enabled object).

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✓

Event ID

Event ID	Objects
pbm_lbuttondown	Window

Arguments

Argument	Description
flags	UnsignedLong by value (the modifier keys and mouse buttons that are pressed). Values are: • 1 — Left mouse button • 2 — Right mouse button • 4 — Shift key • 8 — Ctrl key • 16 — Middle mouse button In the MouseDown event, the left mouse button is always down, so 1 is always summed in the value of *flags*. For an explanation of *flags*, see Syntax 2 of MouseMove on page 245.
xpos	Integer by value (the distance of the pointer from the left edge of the window's workspace in pixels).
ypos	Integer by value (the distance of the pointer from the top of the window's workspace in pixels).

Return codes

Long. Return code choices (specify in a RETURN statement):

0 Continue processing

Examples

Example 1 This code in the MouseDown event displays the window coordinates of the pointer as reported in the *xpos* and *ypos* arguments:

```
sle_2.Text = "Position of Pointer is: " + &
   String(xpos) + "," + String(ypos)
```

Example 2 This code in the MouseDown event checks the value of the flags argument, and reports which modifier keys are pressed in the SingleLineEdit sle_modkey:

```
CHOOSE CASE flags
   CASE 1
      sle_mkey.Text = "No modifier keys pressed"
   CASE 5
      sle_mkey.Text = "SHIFT key pressed"
   CASE 9
      sle_mkey.Text = "CONTROL key pressed"
   CASE 13
      sle_mkey.Text = "SHIFT and CONTROL keys pressed"
END CHOOSE
```

See also

Clicked
MouseMove
MouseUp

MouseMove

The MouseMove event has different arguments for different objects:

Object	See
RichTextEdit control	Syntax 1
Window	Syntax 2

Syntax 1 For RichTextEdit controls

Description

Occurs when the mouse has moved within the RichTextEdit control.

PocketBuilder	✗
PowerBuilder	✓

Event ID

Event ID	Objects
pbm_renmousemove	RichTextEdit

Arguments

None

Return codes

Long. Return code choices (specify in a RETURN statement):

0 Continue processing

Syntax 2 For windows

Description

Occurs when the pointer is moved within the window.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✓

Event ID

Event ID	Objects
pbm_mousemove	Window

Arguments

Argument	Description
flags	UnsignedLong by value (the modifier keys and mouse buttons that are pressed). Values are: • 1 — Left mouse button • 2 — Right mouse button • 4 — Shift key • 8 — Ctrl key • 16— Middle mouse button *Flags* is the sum of all the buttons and keys that are pressed.
xpos	Integer by value (the distance of the pointer from the left edge of the window's workspace in pixels).
ypos	Integer by value (the distance of the pointer from the top of the window's workspace in pixels).

Return codes

Long. Return code choices (specify in a RETURN statement):

0 Continue processing

Usage

Because *flags* is a sum of button and key numbers, you can find out what keys are pressed by subtracting the largest values one by one and checking the value that remains. For example:

- If *flags* is 5, the Shift key (4) and the left mouse button (1) are pressed.
- If *flags* is 14, the Ctrl key (8), the Shift key (4), and the right mouse button (2) are pressed.

This code handles all the buttons and keys (the local boolean variables are initialized to false by default):

```
boolean lb_left_button, lb_right_button
boolean lb_middle_button, lb_Shift_key, lb_control_key
integer li_flags

li_flags = flags
IF li_flags  15 THEN
   // Middle button is pressed
   lb_middle_button = TRUE
   li_flags = li_flags - 16
END IF
```

```
IF li_flags  7 THEN
   // Control key is pressed
   lb_control_key = TRUE
   li_flags = li_flags - 8
END IF

IF li_flags > 3 THEN
   // Shift key is pressed
   lb_Shift_key = TRUE
   li_flags = li_flags - 4
END IF

IF li_flags > 1 THEN
   // Right button is pressed
   lb_lb_right_button = TRUE
   li_flags = li_flags - 2
END IF

IF li_flags = 1 THEN lb_left_button = TRUE
```

Most controls in a window do not capture MouseMove events—the MouseMove event is not mapped by default. If you want the window's MouseMove event to be triggered when the mouse moves over a control, you must map a user-defined event to the pbm_mousemove event for the control. The following code in the control's user-defined MouseMove event triggers the window's MouseMove event:

```
Parent.EVENT MouseMove(0, Parent.PointerX(),
   Parent.PointerY())
```

Examples

This code in the MouseMove event causes a meter OLE custom control to rise and fall continually as the mouse pointer is moved up and down in the window workspace:

```
This.uf_setmonitor(ypos, ole_verticalmeter, &
   This.WorkspaceHeight() )
```

Uf_setmonitor is a window function that scales the pixels to the range of the gauge. It accepts three arguments: the vertical position of the mouse pointer, an OLECustomControl reference, and the maximum range of the mouse pointer for scaling purposes:

```
double ld_gaugemax, ld_gaugemin
double ld_gaugerange, ld_value
```

```
// Ranges for monitor-type control
ld_gaugemax = ocxitem.Object.MaxValue
ld_gaugemin = ocxitem.Object.MinValue
ld_gaugerange = ld_gaugemax - ld_gaugemin

// Horizontal position of mouse within window
ld_value = data * ld_gaugerange / range + ld_gaugemin

// Set gauge
ocxitem.Object.Value = Round(ld_value, 0)

RETURN 1
```

The OLE custom control also has a MouseMove event. This code in that event keeps the gauge responding when the pointer is over the gauge. (You need to pass values for the arguments that are usually handled by the system; the mouse position values are specified in relation to the parent window.) For example:

```
Parent.EVENT MouseMove(0, Parent.PointerX(), &
Parent.PointerY())
```

See also

Clicked
MouseDown
MouseUp

MouseUp

The MouseUp event has different arguments for different objects:

Object	See
RichTextEdit control	Syntax 1
Window	Syntax 2

Syntax 1 For RichTextEdit controls

Description

Occurs when the user releases the left mouse button in a RichTextEdit control.

PocketBuilder	✗
PowerBuilder	✓

Event ID

Event ID	Objects
pbm_renlbuttonup	RichTextEdit

Arguments None

Return codes Long. Return code choices (specify in a RETURN statement):
0 Continue processing

Syntax 2 For windows

Description Occurs when the user releases the left mouse button in an unoccupied area of the window (any area with no visible enabled object).

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✓

Event ID

Event ID	Objects
pbm_lbuttonup	Window

Arguments

Argument	Description
flags	UnsignedLong by value (the modifier keys and mouse buttons that are pressed). Values are: • 1 — Left mouse button • 2 — Right mouse button • 4 — Shift key • 8 — Ctrl key • 16 — Middle mouse button In the MouseUp event, the left mouse button is being released, so 1 is not summed in the value of *flags*. For an explanation of *flags*, see Syntax 2 of MouseMove on page 245.
xpos	Integer by value (the distance of the pointer from the left edge of the window's workspace in pixels).
ypos	Integer by value (the distance of the pointer from the top of the window's workspace in pixels).

Return codes

Long. Return code choices (specify in a RETURN statement):

0 Continue processing

Usage

A Clicked event also occurs when the mouse button is released.

Examples

Example 1 This code in the window's MouseUp event displays in the SingleLineEdit sle_2 the window coordinates of the pointer when the button is released as reported in the *xpos* and *ypos* arguments.

```
sle_2.Text = "Position of Pointer is: " + &
   String(xpos) + "," + String(ypos)
```

Example 2 This code in the window's MouseUp event checks the value of the flags argument and reports which modifier keys are pressed in the SingleLineEdit sle_modkey.

```
CHOOSE CASE flags
   CASE 0
      sle_mkey.Text = "No modifier keys pressed"

   CASE 4
      sle_mkey.Text = "SHIFT key pressed"

   CASE 8
      sle_mkey.Text = "CONTROL key pressed"

   CASE 12
      sle_mkey.Text = "SHIFT and CONTROL keys pressed"

END CHOOSE
```

See also

Clicked
MouseDown
MouseMove

Moved

Description

Occurs when the user moves the scroll box, either by clicking on the arrows or by dragging the box itself.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Event ID

Event ID	Objects
pbm_sbnthumbtrack	HScrollBar, HTrackBar, VScrollBar, VTrackBar

Arguments

Argument	Description
scrollpos	Integer by value (a number indicating position of the scroll box within the range of values specified by the MinPosition and MaxPosition properties)

Return codes

Long. Return code choices (specify in a RETURN statement):

0 Continue processing

Usage

The Moved event updates the Position property of the scroll bar with the value of *scrollpos*.

Examples

This statement in the Moved event displays the new position of the scroll box in a StaticText control:

```
st_1.Text = "Moved " + String(scrollpos)
```

See also

LineDown
LineLeft
LineRight
LineUp
PageDown
PageLeft
PageRight
PageUp

Open

The Open event has different arguments for different objects:

Object	See
Application	Syntax 1
Window	Syntax 2

Syntax 1 For the application object

Description

Occurs when the user starts the application.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Event ID

Event ID	Objects
None	Application

Arguments

Argument	Description
commandline	String by value. Additional arguments are included on the command line after the name of the executable program.

Return codes

None (do not use a RETURN statement)

Usage

This event can establish database connection parameters and open the main window of the application.

There is no way to specify command line values when you are testing your application in the development environment.

Opening the application with command-line arguments at runtime
You can specify command line arguments when you use the Run command from the Start menu or as part of the Target specification when you define a shortcut for your application.

In other events and functions, you can call the CommandParm function to get the command line arguments.

For an example of parsing the string in *commandline*, see CommandParm on page 379.

Examples

This example populates the SQLCA global variable from the application's initialization file, connects to the database, and opens the main window:

```
/* Populate SQLCA from current myapp.ini settings */
SQLCA.DBMS = ProfileString("myapp.ini", "database", &
   "dbms", "")
SQLCA.Database = ProfileString("myapp.ini", &
   "database", "database", "")
SQLCA.Userid = ProfileString("myapp.ini", "database", &
   "userid", "")
SQLCA.DBPass = ProfileString("myapp.ini", "database", &
   "dbpass", "")
SQLCA.Logid = ProfileString("myapp.ini", "database", &
   "logid", "")
SQLCA.Logpass = ProfileString("myapp.ini", &
   "database", "LogPassWord", "")
SQLCA.Servername = ProfileString("myapp.ini", &
   "database", "servername", "")
SQLCA.DBParm = ProfileString("myapp.ini", "database", &
   "dbparm", "")

CONNECT;

IF SQLCA.Sqlcode <> 0 THEN
   MessageBox("Cannot Connect to Database", &
      SQLCA.SQLErrText)
   RETURN
END IF

/* Open MDI frame window */
Open(w_genapp_frame)
```

See also

Close

Syntax 2 For windows

Description

Occurs when a window is opened by one of the Open functions. The event occurs after the window has been opened but before it is displayed.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Event ID

Event ID	Objects
pbm_open	Window

Arguments

None

Return codes

Long. Return code choices (specify in a RETURN statement):

0 Continue processing

Usage

These functions trigger the Open event:

Open
OpenWithParm
OpenSheet
OpenSheetWithParm

When the Open event occurs, the controls on the window already exist (their Constructor events have occurred). In the Open event script, you can refer to objects in the window and affect their appearance or content. For example, you can disable a button or retrieve data for a DataWindow.

Some actions are not appropriate in the Open event, even though all the controls exist. For example, calling the SetRedraw function for a control fails because the window is not yet visible.

Changing the WindowState property
Do not change the WindowState property in the Open event of a window opened as a sheet. Doing so might result in duplicate controls on the title bar. You can change the property in other scripts once the window is open.

When a window is opened, other events occur, such as Constructor for each control in the window, Activate and Show for the window, and GetFocus for the first control in the window's tab order.

Examples

When the window contains a DataWindow control, you can retrieve data for it in the Open event. In this example, values for the transaction object SQLCA have already been set up:

```
dw_1.SetTransObject(SQLCA)
dw_1.Retrieve( )
```

See also

Activate
Constructor
Show

Other

Description

Occurs when a system message occurs that is not a PowerBuilder message.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Event ID

Event ID	Objects
pbm_other	Windows and controls that can be placed in windows

Arguments

Argument	Description
wparam	UnsignedLong by value
lparam	Long by value

Return codes

Long. Return code choices (specify in a RETURN statement):

0 Continue processing

Usage

The Other event is no longer useful, because you can define your own user events. You should avoid using it, because it slows performance while it checks every Windows message.

PageDown

Description

Occurs when the user clicks in the open space below the scroll box.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Event ID

Event ID	Objects
pbm_sbnpagedown	VScrollBar, VTrackBar

Arguments

None

Return codes

Long. Return code choices (specify in a RETURN statement):

0 Continue processing

Usage

When the user clicks in a vertical scroll bar, nothing happens unless you have scripts that change the scroll bar's Position property. For the scroll bar arrows, use the LineUp and LineDown events; for clicks in the scroll bar background above and below the thumb, use the PageUp and PageDown events; for dragging the thumb itself, use the Moved event.

Examples

Example 1 This code in the VScrollBar's PageDown event uses a predetermined paging value stored in the instance variable *ii_pagesize* to change the position of the scroll box (you would need additional code to change the view of associated controls according to the scroll bar position):

```
IF This.Position > &
   This.MaxPosition - ii_pagesize THEN
   This.Position = MaxPosition
ELSE
   This.Position = This.Position + ii_pagesize
END IF
RETURN 0
```

Example 2 This example changes the position of the scroll box by a predetermined page size stored in the instance variable *ii_pagesize* and scrolls forward through a DataWindow control 10 rows for each page:

```
long ll_currow, ll_nextrow

This.Position = This.Position + ii_pagesize
ll_currow = dw_1.GetRow()
ll_nextrow = ll_currow + 10
```

```
dw_1.ScrollToRow(ll_nextrow)
dw_1.SetRow(ll_nextrow)
```

See also

LineDown
PageLeft
PageRight
PageUp

PageLeft

Description

Occurs when the open space to the left of the scroll box is clicked.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Event ID

Event ID	Objects
pbm_sbnpageup	HScrollBar, HTrackBar

Arguments

None

Return codes

Long. Return code choices (specify in a RETURN statement):

0 Continue processing

Usage

When the user clicks in a horizontal scroll bar, nothing happens unless you have scripts that change the scroll bar's Position property. For the scroll bar arrows, use the LineLeft and LineRight events; for clicks in the scroll bar background above and below the thumb, use the PageLeft and Right events; for dragging the thumb itself, use the Moved event.

Examples

This code in the PageLeft event causes the thumb to move left a predetermined page size when the user clicks on the left arrow of the horizontal scroll bar (the page size is stored in the instance variable *ii_pagesize*):

```
IF This.Position < &
This.MinPosition + ii_pagesize THEN
   This.Position = MinPosition
ELSE
   This.Position = This.Position - ii_pagesize
END IF
```

See also

LineLeft
PageDown
PageRight
PageUp

PageRight

Description

Occurs when the open space to the right of the scroll box is clicked.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Event ID

Event ID	Objects
pbm_sbnpagedown	HScrollBar, HTrackBar

Arguments

None

Return codes

Long. Return code choices (specify in a RETURN statement):

0 Continue processing

Usage

When the user clicks in a horizontal scroll bar, nothing happens unless you have scripts that change the scroll bar's Position property:

- For the scroll bar arrows, use the LineLeft and LineRight events.
- For clicks in the scroll bar background above and below the thumb, use the PageLeft and Right event.
- For dragging the thumb itself, use the Moved event.

Examples

This code in the PageRight event causes the thumb to move right when the user clicks on the right arrow of the horizontal scroll bar (the page size is stored in the instance variable *ii_pagesize*):

```
IF This.Position > &
This.MaxPosition - ii_pagesize THEN
   This.Position = MaxPosition
ELSE
   This.Position = This.Position + ii_pagesize
END IF
```

See also

LineRight
PageDown
PageLeft
PageUp

PageUp

Description

Occurs when the user clicks in the open space above the scroll box (also called the *thumb*).

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Event ID

Event ID	Objects
pbm_sbnpageup	VScrollBar, VTrackBar

Arguments

None

Return codes

Long. Return code choices (specify in a RETURN statement):

0 Continue processing

Usage

When the user clicks in a vertical scroll bar, nothing happens unless you have scripts that change the scroll bar's Position property:

- For the scroll bar arrows, use the LineUp and LineDown events.
- For clicks in the scroll bar background above and below the thumb, use the PageUp and PageDown events.
- For dragging the thumb itself, use the Moved event.

Examples

Example 1 This code in the PageUp event causes the thumb to move up when the user clicks on the up arrow of the vertical scroll bar (the page size is stored in the instance variable *ii_pagesize*):

```
IF This.Position < &
This.MinPosition + ii_pagesize THEN
   This.Position = MinPosition
ELSE
   This.Position = This.Position - ii_pagesize
END IF
```

Example 2 This example changes the position of the scroll box by a predetermined page size stored in the instance variable *ii_pagesize* and scrolls backwards through a DataWindow control 10 rows for each page:

```
long ll_currow, ll_prevrow
This.Position = This.Position - ii_pagesize
ll_currow = dw_1.GetRow( )
ll_prevrow = ll_currow - 10
dw_1.ScrollToRow(ll_prevrow)
dw_1.SetRow(ll_prevrow)
```

See also

LineUp
PageDown
PageLeft
PageRight

PictureSelected

Description

Occurs when the user selects a bitmap in the RichTextEdit control by double-clicking it or pressing Enter after clicking it.

PocketBuilder	✗
PowerBuilder	✓

Event ID

Event ID	Objects
pbm_renpictureselected	RichTextEdit

Arguments

None

Return codes

Long. Return code choices (specify in a RETURN statement):

0	Continue processing

PipeEnd

Description

Occurs when pipeline processing is completed.

PocketBuilder	✗
PowerBuilder	✓

Event ID

Event ID	Objects
pbm_pipeend	Pipeline

Arguments None

Return codes Long. Return code choices (specify in a RETURN statement):

0 Continue processing

PipeMeter

Description Occurs during pipeline processing after each block of rows is read or written. The Commit factor specified for the Pipeline in the Pipeline painter determines the size of each block.

PocketBuilder	✗
PowerBuilder	✓

Event ID

Event ID	Objects
pbm_pipemeter	Pipeline

Arguments None

Return codes Long. Return code choices (specify in a RETURN statement):

0 Continue processing

PipeStart

Description Occurs when pipeline processing begins.

PocketBuilder	✗
PowerBuilder	✓

Event ID

Event ID	Objects
pbm_pipestart	Pipeline

Arguments None

Return codes

Long. Return code choices (specify in a RETURN statement):

0	Continue processing

Power

Description

Occurs after notification of a power state change on a mobile device.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✗

Event ID

Event ID	Objects
None	Application

Arguments

Argument	Description
petPowerEventType	UnsignedLong by value. The values are: • 0 — (petHibernate!) WM_Hibernate request made. • 1 — (petTransition!) System power state transition detected. • 2 — (petResume!) Resumed from hibernate. • 4 — (petStatusChange!) Power switched from AC to DC. • 8 — (petInfoChange!) Other power status field change. • 256 — (petSuspendKeyPressed!) Suspend key pressed.
psfPowerStateFlags	UnsignedLong by value. The values are: • 0 — (psfUnknown!) Unknown or not applicable. • 1 — (psfOn!) On state. • 2 — (psfOff!) Full off. • 4 — (psfCritical!) Critical off. • 8 — (psfBoot!) Boot state. • 16 — (psfIdle!) Idle state. • 32 — (psfSuspend!) Suspend state. • 128 — (psfReset!) Reset state. • 256 — (psfUserIdle!) User idle state. • 4096 — (psfPassword!) Password protected state.

Return codes

None.

Usage

This event is not supported on all devices. In particular, it is not supported on any Pocket PC 2002 devices. Other devices might support this event only partially, with the level of support depending on the specific device model.

PrintFooter

Description

Occurs when the footer of a page of the document in the RichTextEdit control is about to be printed.

PocketBuilder	✗
PowerBuilder	✓

Event ID

Event ID	Objects
pbm_renprintfooter	RichTextEdit

Return codes

Long. Return code choices (specify in a RETURN statement):

0	Continue processing
1	Do not print the header for the current page

PrintHeader

Description

Occurs when the header of a page of the document in the RichTextEdit control is about to be printed.

PocketBuilder	✗
PowerBuilder	✓

Event ID

Event ID	Objects
pbm_renprintheader	RichTextEdit

Return codes

Long. Return code choices (specify in a RETURN statement):

0	Continue processing
1	Do not print the header for the current page

ProgressIndex

Description

Reserved for future use. Occurs periodically during synchronization after updates to a synchronization progress bar.

Event ID

Event ID	Objects
None	MLSynchronization, MLSync

Arguments

Argument	Description
progress_idx	Long value representing the progress of the synchronization.
progress_max	Long value indicating the progress limit of the synchronization.

Return codes

None

PropertyChanged

Description

Occurs after the OLE server changes the value of a property of the OLE object.

PocketBuilder	✗
PowerBuilder	✓

Event ID

Event ID	Objects
None	OLE

Return codes

None (do not use a RETURN statement)

PropertyRequestEdit

Description

Occurs when the OLE server is about to change the value of a property of the object in the OLE control.

PocketBuilder	✗
PowerBuilder	✓

Event ID

Event ID	Objects
None	OLE

Return codes

None. Do not use a RETURN statement.

RButtonDown

The RButtonDown event has different arguments for different objects:

Object	See
Controls and windows, except RichTextEdit	Syntax 1
RichTextEdit control	Syntax 2

Syntax 1 For controls and windows, except RichTextEdit

Description

For a window, occurs when the right mouse button is pressed in an unoccupied area of the window (any area with no visible, enabled object). The window event will occur if the cursor is over an invisible or disabled control.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✓

PocketBuilder
On a Pocket PC device, tap and hold the stylus to trigger the RButtonDown event.

For a control, occurs when the right mouse button is pressed on the control.

Event ID

Event ID	Objects
pbm_rbuttondown	Windows and controls that can be placed on a window, except RichTextEdit

Arguments

Argument	Description
flags	UnsignedLong by value (the modifier keys and mouse buttons that are pressed). Values are: • 1 — Left mouse button • 2 — Right mouse button • 4 — Shift key • 8 — Ctrl key • 16 — Middle mouse button In the RButtonDown event, the right mouse button is always pressed, so 2 is always summed in the value of *flags*. For an explanation of *flags*, see Syntax 2 of MouseMove on page 245.
xpos	Integer by value (the distance of the pointer from the left edge of the window's workspace in pixels).
ypos	Integer by value (the distance of the pointer from the top of the window's workspace in pixels).

Return codes

Long. Return code choices (specify in a RETURN statement):

0 Continue processing

Examples

These statements in the RButtonDown script for the window display a pop-up menu at the cursor position. Menu4 was created in the Menu painter and includes a menu called m_language. Menu4 is not the menu for the active window and therefore needs to be created. *NewMenu* is an instance of Menu4 (datatype Menu4):

```
Menu4 NewMenu
NewMenu = CREATE Menu4
NewMenu.m_language.PopMenu(xpos, ypos)
```

In a Multiple Document Interface (MDI) application, the arguments for PopMenu need to specify coordinates relative to the MDI frame:

```
NewMenu.m_language.PopMenu( &
   w_frame.PointerX(), w_frame.PointerY())
```

See also

Clicked

Syntax 2 For RichTextEdit controls

Description

Occurs when the user presses the right mouse button on the RichTextEdit control and the control's PopMenu property is set to false.

PocketBuilder	✗
PowerBuilder	✓

Event ID

Event ID	Objects
pbm_renrbuttondown	RichTextEdit

Arguments

None

Return codes

Long. Return code choices (specify in a RETURN statement):

0	Continue processing

RButtonUp

Description

Occurs when the right mouse button is released.

PocketBuilder	✗
PowerBuilder	✓

Event ID

Event ID	Objects
pbm_renrbuttonup	RichTextEdit

Arguments

None

Return codes

Long. Return code choices (specify in a RETURN statement):

0	Continue processing
1	Prevent processing

RemoteExec

Description Occurs when a DDE client application has sent a command.

PocketBuilder	✗
PowerBuilder	✓

Event ID

Event ID	Objects
pbm_ddeexecute	Window

Arguments None

Return codes Long. Return code choices (specify in a RETURN statement):

0 Continue processing

RemoteHotLinkStart

Description Occurs when a DDE client application wants to start a hot link.

PocketBuilder	✗
PowerBuilder	✓

Event ID

Event ID	Objects
pbm_ddeadvise	Window

Arguments None

Return codes Long. Return code choices (specify in a RETURN statement):

0 Continue processing

RemoteHotLinkStop

Description Occurs when a DDE client application wants to end a hot link.

PocketBuilder	✗
PowerBuilder	✓

Event ID

Event ID	Objects
pbm_ddeunadvise	Window

Arguments None

Return codes Long. Return code choices (specify in a RETURN statement):

0 Continue processing

RemoteRequest

Description Occurs when a DDE client application requests data.

PocketBuilder	✗
PowerBuilder	✓

Event ID

Event ID	Objects
pbm_dderequest	Window

Arguments None

Return codes Long. Return code choices (specify in a RETURN statement):

0 Continue processing

RemoteSend

Description Occurs when a DDE client application has sent data.

PocketBuilder	✗
PowerBuilder	✓

Event ID

Event ID	Objects
pbm_ddepoke	Window

Arguments None

Return codes

Long. Return code choices (specify in a RETURN statement):

0 Continue processing

Rename

Description

Occurs when the server application notifies the control that the object has been renamed.

PocketBuilder	✗
PowerBuilder	✓

Event ID

Event ID	Objects
pbm_omnrename	OLE

Arguments

None

Return codes

Long. Return code: Ignored

Resize

Description

Occurs when the user or a script opens or resizes the client area of a window or DataWindow control.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Event ID

Event ID	Objects
pbm_dwnresize	DataWindow
pbm_size	Window

Arguments

Argument	Description
sizetype	UnsignedLong by value. The values are: • 0 — (SIZE_RESTORED) The window or DataWindow has been resized, but it was not minimized or maximized. The user might have dragged the borders or a script might have called the Resize function. • 1 — (SIZE_MINIMIZED) The window or DataWindow has been minimized. • 2 — (SIZE_MAXIMIZED) The window or DataWindow has been maximized. • 3 — (SIZE_MAXSHOW) This window is a pop-up window and some other window in the application has been restored to its former size (does not apply to DataWindow controls). • 4 — (SIZE_MAXHIDE) This window is a pop-up window and some other window in the application has been maximized (does not apply to DataWindow controls).
newwidth	Integer by value (the width of the client area of a window or DataWindow control in PowerBuilder units).
newheight	Integer by value (the height of the client area of a window or DataWindow control in PowerBuilder units).

Return codes

Long. Return code choices (specify in a RETURN statement):

0 Continue processing

RightClicked

The RightClicked event has different arguments for different objects:

Object	See
ListView and Tab control	Syntax 1
TreeView control	Syntax 2

Syntax 1 For ListView and Tab controls

Description

Occurs when the user clicks the right mouse button on the ListView control or the tab portion of the Tab control.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✓

PocketBuilder applications
In PocketBuilder applications, tap and hold the stylus to trigger the RightClicked event.

Event ID

Event ID	Objects
pbm_lvnrclicked	ListView
pbm_tcnrclicked	Tab

Arguments

Argument	Description
index	Integer by value (the index of the item or tab the user clicked)

Return codes

Long. Return code choices (specify in a RETURN statement):

0 Continue processing

Usage

When the user clicks in the display area of the Tab control, the tab page user object gets an RButtonDown event rather than a RightClicked event for the Tab control.

Examples

This example for the RightClicked event of a ListView control displays a pop-up menu when the user clicks the right mouse button:

```
// Declare a menu variable of type m_main
m_main m_lv_popmenu
// Create an instance of the menu variable
m_lv_popmenu = CREATE m_main
// Display menu at pointerposition
m_lv_popmenu.m_entry.PopMenu(Parent.PointerX(), &
   Parent.PointerY())
```

See also

Clicked
RightDoubleClicked

Syntax 2 For TreeView controls

Description

Occurs when the user clicks the right mouse button on the TreeView control.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✓

PocketBuilder applications
In PocketBuilder applications, tap and hold the stylus to trigger the RightClicked event.

Event ID

Event ID	Objects
pbm_tvnrclicked	TreeView

Arguments

Argument	Description
handle	Long by value (the handle of the item the user clicked)

Return codes

Long. Return code choices (specify in a RETURN statement):

0 Continue processing

Examples

This example for the RightClicked event of a TreeView control displays a pop-up menu when the user clicks the right mouse button:

```
// Declare a menu variable of type m_main
m_main m_tv_popmenu

// Create an instance of the menu variable
m_tv_popmenu = CREATE m_main

// Display menu at pointer position
m_tv_popmenu.m_entry.PopMenu(Parent.PointerX(), &
   Parent.PointerY())
```

See also

Clicked
RightDoubleClicked

RightDoubleClicked

The RightDoubleClicked event has different arguments for different objects:

Object	See
ListView and Tab control	Syntax 1
TreeView control	Syntax 2

Syntax 1 For ListView and Tab controls

Description

Occurs when the user double-clicks the right mouse button on the ListView control or the tab portion of the Tab control.

PocketBuilder	✗
PowerBuilder	✓

Event ID

Event ID	Objects
pbm_lvnrdoubleclicked	ListView
pbm_tcnrdoubleclicked	Tab

Return codes

Long. Return code choices (specify in a RETURN statement):

0 Continue processing

Syntax 2 For TreeView controls

Description

Occurs when the user double-clicks the right mouse button on the TreeView control.

PocketBuilder	✗
PowerBuilder	✓

Event ID

Event ID	Objects
pbm_tvnrdoubleclicked	TreeView

Return codes

Long. Return code choices (specify in a RETURN statement):

0 Continue processing

Save

Description

Occurs when the server application notifies the control that the data has been saved.

PocketBuilder	✗
PowerBuilder	✓

Event ID

Event ID	**Objects**
pbm_omnsave	OLE

Arguments

None

Return codes

Long. Return code: Ignored

SaveObject

Description

Occurs when the server application saves the object in the control.

PocketBuilder	✗
PowerBuilder	✓

Event ID

Event ID	Objects
pbm_omnsaveobject	OLE

Arguments

None

Return codes

Long. Return code: Ignored

ScanTriggered

Description

Occurs when a scan operation is started.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✗

Event ID

Event ID	Objects
None	BarcodeScanner objects

Arguments

None.

Return codes

None.

Usage

Use this event with the ScanNoWait function and implement as an asynchronous (continuous) scan.

Examples

The following code in the ScanTriggered event implements continuous scanning:

```
// Bar code trigger
// A scan event (typically read) has occured
int    iRet
int    itmp
string stmp
```

```
lb_res.AddItem( "==== Scan Triggered ====" )
lb_res.AddItem("Data: " + this.ScannerName )
iRet = this.RetrieveData()
lb_res.AddItem("RetrieveData: " + string(iRet) )

// ** Display the status **
choose case iRet
    case 1
       lb_res.AddItem("*SUCCESS*")
    case -9
       // common
       lb_res.AddItem("*Incorrect State (aborted?)")
    case -13
       // common
       lb_res.AddItem("*Timeout (benign)")
    case -12
       // common
       lb_res.AddItem("*Read Cancelled")
    // ** and the rare errors **
    case -1
       lb_res.AddItem("*ERR - General")
    case -8
       lb_res.AddItem("*ERR - Buffer Allocation")
    case -10
       lb_res.AddItem("*ERR - Device")
    case -11
       lb_res.AddItem("*ERR - Read Pending")
    case else
       lb_res.AddItem("*ERR - Other")
end choose

// ** Display the data **
if iRet = 1 then
    // Data:
    stmp = this.ScannedData
    lb_res.AddItem("Data: " + stmp )
    // Symbology:
    itmp = this.ScannedSymbology
    stmp = this.Decodername(itmp)
    lb_res.AddItem("Symbology: " + string(itmp) + " : "&
      + stmp )
    // TimeStamp:
    stmp = STRING( this.ScannedTimeStamp, "hh:mm:ss" )
    lb_res.AddItem("TimeStamp: " + stmp )
end if
```

```
// ** Continue? **
if cbx_rearm.checked then
    iRet = this.ScanNoWait()
    lb_res.AddItem("ScanNoWait: " + string(iRet) )
end if

lb_res.SelectItem( lb_res.totalitems() )
```

See also ScanNoWait

Selected

Description Occurs when the user highlights an item on the menu using the arrow keys or the mouse, without choosing it to be executed.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Event ID

Event ID	Objects
None	Menu

Arguments None

Return codes None. (Do not use a RETURN statement.)

Usage You can use the Selected event to display MicroHelp for the menu item (PowerBuilder only). One way to store the Help text is in the menu item's Tag property.

Examples This example uses the tag value of the current menu item to display Help text. The function wf_SetMenuHelp takes the text passed (the tag) and assigns it to a MultiLineEdit control. A Timer function and the Timer event are used to clear the Help text.

This code in the Selected event calls the function that sets the text:

```
w_test.wf_SetMenuHelp(This.Tag)
```

This code for the wf_SetMenuHelp function sets the text in the MultiLineEdit mle_menuhelp; its argument is called *menuhelpstring*:

```
mle_menuhelp.Text = menuhelpstring
Timer(4)
```

This code in the Timer event clears the Help text and stops the timer:

```
w_test.wf_SetMenuHelp("")
Timer(0)
```

See also

Clicked

SelectionChanged

The SelectionChanged event has different arguments for different objects:

Object	See
DropDownListBox, DropDownPictureListBox, ListBox, PictureListBox controls	Syntax 1
Tab control	Syntax 2
TreeView control	Syntax 3

Syntax 1 For Listboxes

Description

Occurs when an item is selected in the control.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Event ID

Event ID	Objects
pbm_cbnselchange	DropDownListBox, DropDownPictureListBox
pbm_lbnselchange	ListBox, PictureListBox

Arguments

Argument	Description
index	Integer by value (the index of the item that has become selected)

Return codes

Long. Return code choices (specify in a RETURN statement):

0 Continue processing

Usage

For DropDownListBoxes, the SelectionChanged event applies to selections in the drop-down portion of the control, not the edit box.

The SelectionChanged event occurs when the user clicks on any item in the list, even if it is the currently selected item. When the user makes a selection using the mouse, the Clicked (and if applicable the DoubleClicked event) occurs after the SelectionChanged event.

Examples

This example is for the lb_value ListBox in the window w_graph_sheet_with_list (in the PowerBuilder Examples application). When the user chooses values, they are graphed as series in the graph gr_1. The MultiSelect property for the ListBox is set to true, so *index* has no effect. The script checks all the items to see if they are selected:

```
integer itemcount,i,r
string ls_colname

gr_1.SetRedraw(FALSE)

// Clear out categories, series and data from graph
gr_1.Reset(All!)

// Loop through all selected values and
// create as many series as the user specified
FOR i = 1 to lb_value.TotalItems()
   IF lb_value.State(i) = 1 THEN
      ls_colname = lb_value.Text(i)

      // Call window function to set up the graph
      wf_set_a_series(ls_colname, ls_colname, &
      lb_category.text(1))
   END IF
NEXT
gr_1.SetRedraw(TRUE)
```

See also

Clicked

Syntax 2 For Tab controls

Description

Occurs when a tab is selected.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✓

Event ID

Event ID	Objects
pbm_tcnselchanged	Tab

Arguments

Argument	Description
oldindex	Integer by value (the index of the tab that was previously selected)
newindex	Integer by value (the index of the tab that has become selected)

Return codes

Long. Return code choices (specify in a RETURN statement):

0 Continue processing

Usage

The SelectionChanged event occurs when the Tab control is created and the initial selection is set.

See also

Clicked
SelectionChanging

Syntax 3 For TreeView controls

Description

Occurs when the item is selected in a TreeView control.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Event ID

Event ID	Objects
pbm_tvnselchanged	TreeView

Arguments

Argument	Description
oldhandle	Long by value (the handle of the previously selected item)
newhandle	Long by value (the handle of the currently selected item)

Return codes

Long. Return code choices (specify in a RETURN statement):

0	Continue processing

Usage

The SelectionChanged event occurs after the SelectionChanging event.

Examples

This example tracks items in the SelectionChanged event:

```
TreeViewIteml_tvinew, l_tviold

// get the treeview item that was the old selection
This.GetItem(oldhandle, l_tviold)

// get the treeview item that is currently selected
This.GetItem(newhandle, l_tvinew)

// Display the labels for the two items in sle_get
sle_get.Text = "Selection changed from " &
   + String(l_tviold.Label) + " to " &
   + String(l_tvinew.Label)
```

See also

Clicked
SelectionChanging

SelectionChanging

The SelectionChanging event has different arguments for different objects:

Object	See
Tab control	Syntax 1
TreeView control	Syntax 2

Syntax 1 For Tab controls

Description

Occurs when another tab is about to be selected.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✓

Event ID

Event ID	Objects
pbm_tcnselchanging	Tab

Arguments

Argument	Description
oldindex	Integer by value (the index of the currently selected tab)
newindex	Integer by value (the index of the tab that is about to be selected)

Return codes

Long. Return code choices (specify in a RETURN statement):

0	Allow the selection to change
1	Prevent the selection from changing

Usage

Use the SelectionChanging event to prevent the selection from changing or to do processing for the newly selected tab page before it becomes visible. If CreateOnDemand is true and this is the first time the tab page is selected, the controls on the page do not exist yet, and you cannot refer to them in the event script.

Examples

When the user selects a tab, this code sizes the DataWindow control on the tab page to match the size of another DataWindow control. The resizing happens before the tab page becomes visible. This example is from tab_uo in the w_phone_dir window in the PowerBuilder Examples:

```
u_tab_dirluo_Tab
luo_Tab = This.Control[newindex]
```

```
luo_Tab.dw_dir.Height = dw_list.Height
luo_Tab.dw_dir.Width = dw_list.Width
```

See also

Clicked
SelectionChanged

Syntax 2 For TreeView controls

Description

Occurs when the selection is about to change in the TreeView control.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Event ID

Event ID	Objects
pbm_tvnselchanging	TreeView

Arguments

Argument	Description
oldhandle	Long by value (the handle of the currently selected item)
newhandle	Long by value (the handle of the item that is about to be selected)

Return codes

Long. Return code choices (specify in a RETURN statement):

0	Allow the selection to change
1	Prevent the selection from changing

Usage

The SelectionChanging event occurs before the SelectionChanged event.

Examples

This example displays the status of changing TreeView items in a SingleLineEdit:

```
TreeViewItem l_tvinew, l_tviold

// Get TreeViewItem that was the old selection
This.GetItem(oldhandle, l_tviold)

// Get TreeViewItem that is currently selected
This.GetItem(newhandle, l_tvinew)
```

```
//Display the labels for the two items in display
sle_status.Text = "Selection changed from " &
   + String(l_tviold.Label) + " to " &
   + String(l_tvinew.Label)
```

See also

Clicked
SelectionChanged

Show

Description

Occurs just before the window is displayed.

PocketBuilder	✗
PowerBuilder	✓

Event ID

Event ID	Objects
pbm_showwindow	Window

Arguments

Argument	Description
show	Boolean by value (whether the window is being shown). The value is always true.
status	Long by value (the status of the window). Values are: • 0 — The current window is the only one affected. • 1 — The window's parent is also being minimized or a pop-up window is being hidden. • 3 — The window's parent is also being displayed or maximized or a pop-up window is being shown.

Return codes

Long. Return code choices (specify in a RETURN statement):

0 Continue processing

Usage

The Show event occurs when the window is opened.

See also

Activate
Hide
Open

SipUp

Description

Occurs when the Soft Input Panel (SIP) is opened.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✗

Event ID

Event ID	Objects
pbm_sipup	Window

Arguments

Argument	Description
flags	UnsignedLong by value. Values are: • 0 — The SIP is off or not visible • 1 — The SIP is visible • 2 — The SIP is docked • 4 — The SIP is locked and the user cannot change its visibility *Flags* is the sum of all SIP states and statuses.

Return codes

Long. Return code choices (specify in a RETURN statement):

0 Continue processing

Usage

Because *flags* is a sum, you can determine the SIP state and status by subtracting the largest values one by one and checking the value that remains. For example:

- If *flags* is 4, the SIP is locked (4), but not docked or visible.
- If *flags* is 5, the SIP is locked (4) and visible (1), but not docked.
- If *flags* is 7, the SIP is locked (4), docked (2), and visible (1).

Examples

In the window's SipUp event, this code returns the SIP type:

```
SIPIMType sType
sType = GetSIPType()
```

See also

SipDown

SipDown

Description

Occurs when the Soft Input Panel (SIP) is closed.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✗

Event ID

Event ID	Objects
pbm_sipdown	Window

Arguments

Argument	Description
flags	UnsignedLong by value. Values are: • 0 — The SIP is off or not visible • 1 — The SIP is visible • 2 — The SIP is docked • 4 — The SIP is locked and the user cannot change its visibility *Flags* is the sum of all SIP states and statuses.

Return codes

Long. Return code choices (specify in a RETURN statement):

0 Continue processing

Usage

Because *flags* is a sum, you can determine the SIP state and status by subtracting the largest values one by one and checking the value that remains. For example:

- If *flags* is 4, the SIP is locked (4), but not docked or visible.
- If *flags* is 5, the SIP is locked (4) and visible (1), but not docked.
- If *flags* is 7, the SIP is locked (4), docked (2), and visible (1).

Examples

In the window's SipDown event, this code gets the coordinates of the window and displays them in a multiline edit box:

```
String strDisplay=""
int rc
long left = 0, top = 0, right = 0, bottom = 0
rc = GetDeskRect(left, top, right, bottom)
```

```
strDisplay +=("Desk RECT:~r~n~t Left = " +string(left)&
   +"~r~n~t Top=" + String(top) + "~r~n~t Right = " &
   + String(right)+ "~r~n~t Bottom = " +
String(bottom))mle_1.text = strDisplay
```

See also

SipUp

Snapped

Description

The Snapped event occurs after an image has been captured by a digital camera device.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✗

Event ID

Event ID	Objects
None	Camera

Arguments

Argument	Description
filename	String by value. The value passed in this argument is the name of the file that stores the snapped image.

Return codes

None (do not use a RETURN statement)

Usage

Coding this event is particularly useful for PocketBuilder applications using the HTC camera. The HTC camera uses IA Camera Wizard software to capture images. (This software can be installed with the camera on the Windows CE device.) The IA Camera Wizard captures the images, and is responsible for notifying the PocketBuilder application of the capture through the Snapped event. It passes back the name of the file containing the image in the *filename* argument. Since the wizard takes care of the image capturing, the image-capturing functions on the camera object are not used.

You can also code this event for other camera devices supported by PocketBuilder, such as the HP and VEO digital cameras. For these cameras, the value in the *filename* argument for the Snapped event is the value that you assign in the CaptureImage function call on the current Camera object.

Examples

The following code in a Snapped event adds notification on a new line in a multiline edit box that an image has been captured:

```
mle_1.text = mle_1.text + "~r~npicture file name is "&
    + filename + "."
```

Sort

The Sort event has different arguments for different objects:

Object	See
ListView control	Syntax 1
TreeView control	Syntax 2

Syntax 1 For ListView controls

Description

Occurs for each comparison when the ListView is being sorted.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Event ID

Event ID	Objects
pbm_lvnsort	ListView

Arguments

Argument	Description
index1	Integer by value (the index of one item being compared during a sorting operation)
index2	Integer by value (the index of the second item being compared)
column	Integer by value (the number of the column containing the items being sorted)

Return codes

Long. Return code choices (specify in a RETURN statement):

-1	*index1* is less than *index2*
0	*index1* is equal to *index2*
1	*index1* is greater than *index2*

Usage

The Sort event allows you to fine-tune the sort order of the items being sorted. You can examine the properties of each item and tell the Sort function how to sort them by selecting one of the return codes.

You typically use the Sort event when you want to sort ListView items based on multiple criteria such as a PictureIndex and Label.

The Sort event occurs if you call the Sort event, or when you call the Sort function using the UserDefinedSort! argument.

Examples

This example sorts ListView items according to PictureIndex and Label sorting by PictureIndex first, and then by label:

```
ListViewItem lvi, lvi2

This.GetItem(index1, lvi)
This.GetItem(index2, lvi2)

IF lvi.PictureIndex > lvi2.PictureIndex THEN
   RETURN 1
ELSEIF lvi.PictureIndex < lvi2.PictureIndex THEN
   RETURN -1
ELSEIF lvi.label > lvi2.label THEN
   RETURN 1
ELSEIF lvi.label < lvi2.label THEN
   RETURN -1
ELSE
   RETURN 0
END IF
```

Syntax 2 For TreeView controls

Description

Occurs for each comparison when the TreeView is being sorted.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Event ID

Event ID	Objects
pbm_tvnsort	TreeView

Arguments

Argument	Description
handle1	Long by value (the handle of one item being compared during a sorting operation)
handle2	Long by value (the handle of the second item being compared)

Return codes

Long. Return code choices (specify in a RETURN statement):

-1	*handle1* is less than *handle2*
0	*handle1* is equal to *handle2*
1	*handle1* is greater than *handle2*

Usage

The Sort event allows you to fine-tune the sort order of the items being sorted. You can examine the properties of each item and tell the Sort function how to sort them by selecting one of the return codes.

You typically use the Sort event when you want to sort TreeView items based on multiple criteria such as a PictureIndex and Label.

The Sort event occurs if you call the Sort event, or when you call the Sort function using the UserDefinedSort! argument.

Examples

This example sorts TreeView items according to PictureIndex and Label sorting by PictureIndex first, then by label:

```
TreeViewItem tvi, tvi2

This.GetItem(handle1, tvi)
This.GetItem(handle2, tvi2)

IF tvi.PictureIndex > tvi2.PictureIndex THEN
   RETURN 1
ELSEIF tvi.PictureIndex < tvi2.PictureIndex THEN
   RETURN -1
ELSEIF tvi.Label > tvi2.Label THEN
   RETURN 1
ELSEIF tvi.Label < tvi2.Label THEN
   RETURN -1
ELSE
   RETURN 0
END IF
```

SyncPreview

Description

Reserved for future use. Returns generated dbmlsync command arguments immediately prior to launching the synchronization process.

Event ID

Event ID	Objects
None	MLSynchronization, MLSync

Arguments

Argument	Description
command_args	String passed by reference that includes dbmlsync command arguments for launching the synchronization process.

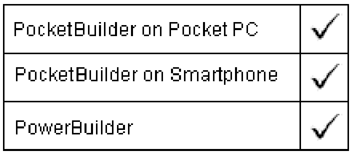
Return codes

None

SystemError

Description

Occurs when a serious execution time error occurs (such as trying to open a nonexistent window) if the error is not handled in a try-catch block.



PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Event ID

Event ID	Objects
None	Application

Arguments

None

Return codes

None. (Do not use a RETURN statement.)

Usage

If there is no script for the SystemError event, PocketBuilder displays a message box with the PocketBuilder error number and error message text.

For errors involving external objects and DataWindows, you can handle the error in the ExternalException or Error events and prevent the SystemError event from occurring. The ExternalException and Error events are maintained for backward compatibility.

You can prevent the SystemError event from occurring by handling errors in try-catch blocks. Well-designed exception-handling code gives application users a better chance to recover from error conditions and run the application without interruption. For information about exception handling, see the *Resource Guide*.

When a SystemError event occurs, your current script terminates and your system might become unstable. It is generally not a good idea to continue running the application, but you can use the SystemError event script to clean up and disconnect from the DBMS before closing the application.

Examples

This statement in the SystemError event halts the application immediately:

```
HALT CLOSE
```

See also

Error
ExternalException
TRY...CATCH...FINALLY...END TRY

SystemKey

Description

Occurs when the insertion point is not in a line edit, and the user presses the Alt key (alone or with another key).

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✓

Event ID

Event ID	Objects
pbm_syskeydown	Window

Arguments

Argument	Description
key	KeyCode by value. A value of the KeyCode enumerated datatype indicating the key that was pressed, for example, KeyA! or KeyF1!.
keyflags	UnsignedLong by value (the modifier keys that were pressed with the key). The only modifier key is the Shift key.

Return codes

Long. Return code choices (specify in a RETURN statement):

0 Continue processing

Usage

Pressing the Ctrl key prevents the SystemKey event from firing when the Alt key is pressed.

Examples

This example displays the name of the key that was pressed with the Alt key:

```
string ls_key

CHOOSE CASE key

CASE KeyF1!
   ls_key = "F1"
CASE KeyA!
   ls_key = "A"
CASE KeyF2!
   ls_key = "F2"
END CHOOSE
```

This example causes a beep if the user presses Alt+Shift+F1.

```
IF keyflags = 1 THEN
   IF key = KeyF1 THEN
      Beep(1)
   END IF
END IF
```

See also

Key

Timer

Description

Occurs when a specified number of seconds elapses after the Start or Timer function has been called.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Event ID

Event ID	Objects
pbm_timer	Timing or Window

Arguments

None

Return codes

Long. Return code choices (specify in a RETURN statement):

0	Continue processing

Examples

These examples show how to use a timing object's Timer event and a window's Timer event.

Using a timing object This example uses a timing object to refresh a list of customers retrieved from a database at specified intervals. The main window of the application, w_main, contains a DataWindow control displaying a list of customers and two buttons, Start Timer and Retrieve. The window's Open event connects to the database:

```
CONNECT using SQLCA;

IF sqlca.sqlcode <> 0 THEN
   MessageBox("Database Connection", &
      sqlca.sqlerrtext)
END IF
```

The following code in the clicked event of the Start Timer button creates an instance of a timing object, nvo_timer, and opens a response window to obtain a timing interval. Then, it starts the timer with the specified interval:

```
MyTimer = CREATE nvo_timer
open(w_interval)
MyTimer.Start(d_interval)

MessageBox("Timer", "Timer Started. Interval is " &
   + string(MyTimer.interval) + " seconds")
```

In the timing object's Constructor event, the following code creates an instance of a datastore:

```
ds_datastore = CREATE datastore
```

The timing object's Timer event calls an object-level function called refresh_custlist that refreshes the datastore. This is the code for refresh_custlist:

```
long ll_rowcount

ds_datastore.dataobject = "d_customers"
ds_datastore.SetTransObject (SQLCA)
ll_rowcount = ds_datastore.Retrieve()

RETURN ll_rowcount
```

The Retrieve button on w_main simply shares the data from the DataStore with the DataWindow control:

```
ds_datastore.ShareData(dw_1)
```

Using a window object This example causes the current time to be displayed in a StaticText control in a window. Calling Timer in the window's Open event script starts the timer. The script for the Timer event refreshes the displayed time.

In the window's Open event script, this code displays the time initially and starts the timer:

```
st_time.Text = String(Now(), "hh:mm")
Timer(60)
```

In the window's Timer event, which is triggered every minute, this code displays the current time in the StaticText st_time:

```
st_time.Text = String(Now(), "hh:mm")
```

ToolbarMoved

Description

Occurs in an MDI frame window when the user moves any FrameBar or SheetBar.

PocketBuilder	✗
PowerBuilder	✓

Event ID

Event ID	Objects
pbm_tbnmoved	Window

Arguments

None

Return codes

Long. Return code choices (specify in a RETURN statement):

0	Continue processing

UploadAck

Description

Reserved for future use. Occurs on completion of upload processing.

Event ID

Event ID	Objects
None	MLSynchronization, MLSync

Arguments

Argument	Description
uploadack_status	String indicating the status returned by MobiLink to the remote after the upload stream is processed.

Return codes None

ViewChange

Description Occurs when the server application notifies the control that the view shown to the user has changed.

PocketBuilder	✗
PowerBuilder	✓

Event ID

Event ID	Objects
pbm_omnviewchange	OLE

Arguments None

Return codes Long. Return code: Ignored

WaitForUploadAck

Description Reserved for future use. Occurs when the synchronization process starts a new waiting period for upload acknowledgement.

Event ID

Event ID	Objects
None	MLSynchronization, MLSync

Arguments None

Return codes None

WarningMessage

Description

Reserved for future use. Occurs on display of a warning message.

Event ID

Event ID	Objects
None	MLSynchronization, MLSync

Arguments

Argument	Description
warnmsg	Read-only string containing the text of the warning message returned from the synchronization server.

Return codes

None

CHAPTER 10 PowerScript Functions

About this chapter

This chapter provides syntax, descriptions, and examples for PowerScript functions.

Contents

The functions are listed alphabetically.

See also

For information about functions that apply to DataWindows or DataStores, see also the *DataWindow Reference*. Methods that apply to DataWindows, but not to other PocketBuilder controls, are listed only in the *DataWindow Reference*.

Abs

Description

Calculates the absolute value of a number.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

Abs (*n*)

Argument	Description
n	The number for which you want the absolute value

Return value

The datatype of *n*. Returns the absolute value of *n*. If *n* is null, Abs returns null.

Examples

All these statements set *num* to 4:

```
integer i, num

i = 4
num = Abs(i)
num = Abs(4)
num = Abs(+4)
num = Abs(-4)
```

This statement returns 4.2:

```
Abs(-4.2)
```

See also

Abs method for DataWindows in the *DataWindow Reference* or online Help

AcceptCall

Description

Accepts a new incoming call.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✗

Applies to

PhoneCall objects

Syntax

objectname.AcceptCall ( )

Argument	Description
objectname	The name of the PhoneCall object that will accept a call.

Return value

Integer. Returns 1 for success and a negative value if an error occurs.

Examples

In the following example, the *g_phInit* global variable is set to 1 in the pcall_1 object's constructor. If the call has been initialized, the AcceptCall function is called. The End Call button is enabled and the New Call button is disabled:

```
// Global variable: Long g_phInit = 0
integer li_ret
if ( g_phInit > 0) then
   li_ret = pcall_1.AcceptCall()
   // enable buttons
   cb_endcall.enabled = true
   cb_newcall.enabled = false
else
   sle_1.text = "Call not initialized"
end if
```

See also

AllowReceivingCalls
DropCall
MakeCall
SetHold
SetMute
SetRingTone

ACos

Description

Calculates the arccosine of an angle.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

ACos (*n*)

Argument	Description
n	The ratio of the lengths of two sides of a triangle for which you want a corresponding angle (in radians). The ratio must be a value between -1 and 1.

Return value

Double. Returns the arccosine of *n*.

Examples

This statement returns 0:

```
ACos(1)
```

This statement returns 3.141593 (rounded to six places):

```
ACos(-1)
```

This statement returns 1.000000 (rounded to six places):

```
ACos(.540302)
```

This code in the Clicked event of a button catches a runtime error that occurs when an arccosine is taken for a user-entered value—passed in a variable—that is outside of the permitted range:

```
Double ld_num
ld_num = Double (sle_1.text)

TRY
sle_2.text = string (acos (ld_num))
CATCH (runtimeerror er)
   MessageBox("Runtime Error", er.getmessage())
END TRY
```

See also

Cos
ASin
ATan
ACos method for DataWindows in the *DataWindow Reference* or online Help

Activate

Description

Activates the object in an OLE container, allowing the user to work with the object using the server's commands.

PocketBuilder	✗
PowerBuilder	✓

Applies to

OLE controls and OLE DWObjects (objects within a DataWindow object that is within a DataWindow control)

Syntax

objectref.**Activate** (*activationtype*)

Return value

Integer. Returns 0 if it succeeds and a negative value if an error occurs.

Add

Description

Adds an appointment, contact, or task as a Pocket Outlook entry.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✗

Applies to

POOM objects

Syntax

Integer *objectname*.Add (*entity*)

Argument	Description
objectname	The name of the POOM object
entity	Entity of type POOMAppointment, POOMContact, or POOMTask

Return value

Integer. Returns 1 for success and one of the following negative values if an error occurs:

-1	Unspecified error
-2	Cannot connect to the repository or a required internal subobject failed to connect to the repository
-3	Cannot log in to the repository
-4	Incorrect input argument
-5	Action cannot be performed
-6	The object identifier (OID) is not in the repository
-7	Feature is not implemented yet
-8	No matching entries found for the criteria

Usage

A user must be logged in to a POOM object to add an appointment, contact, or task. For a POOMTask object, the StartDate property must be set before you call Add. The Body and BodyInk properties cannot be set until after the new object has been added to the repository, but their values are updated in the repository implicitly when either value is set.

Examples

The following example adds an appointment to the depository, adds body text, and displays the appointment in the Pocket PC Calendar:

```
// Global variable: g_poom
integer li_rc
POOMAppointment appt
DateTime  dt
```

```
Date ld_date
Time lt_time

appt = CREATE POOMAppointment
appt.Subject = "All Hands"
appt.Location = "Auditorium"

// get the start and end times from EditMasks
ld_date = Date(em_startdate.Text)
lt_time = Time(em_starttime.Text)
dt = DateTime(ld_date, lt_time)
appt.appointmentStart = dt

ld_date = Date(em_enddate.Text)
lt_time = Time(em_endtime.Text)
dt = DateTime(ld_date, lt_time)
li_rc = g_poom.Add( appt )

// Now add the body of the appointment
appt.Body = "Agenda: ~r~n Quarterly results " &
    + "~r~n Success stories" &
    + "~r~n Organizational changes"

// Display the appointment
appt.display()
```

See also

Login
Remove

AddCategory

Description

Adds a new category to the category axis of a graph. AddCategory is for a category axis whose datatype is string.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Graph controls in windows and user objects. Does not apply to graphs within DataWindow objects because their data comes directly from the DataWindow.

Syntax

controlname.**AddCategory** (*categoryname*)

Argument	Description
controlname	The name of the graph to which you want to add a category.
categoryname	A string whose value is the name of the category you want to add to *controlname*. The category will appear as a label on the category axis.

Return value

Integer. Returns the number assigned to the category if it succeeds. If *categoryname* already exists as a label on the category axis, AddCategory returns the number of the existing category. Returns -1 if an error occurs. If any argument's value is null, AddCategory returns null.

Usage

AddCategory adds a category to the end of the category axis. The category becomes an empty slot in each series to which you can assign a data point. A tick mark exists on the category axis for all the categories associated with the graph.

When the datatype of the category axis is string, you can specify the empty string ("") as the category name. However, because category names must be unique, there can be only one category with that name. Category names are unique if they have different capitalization.

To add categories when the axis datatype is date, DateTime, number, or time, use InsertCategory. To insert a category in the middle of a series, use InsertCategory. You can also use InsertCategory to add a category to the end of a series, as AddCategory does, but it requires an additional argument to do so.

To add data to a series in the graph, use the AddData or InsertData function. You can add a data value and put it in a new category, or you can add or change data in an existing category. To add a series to the graph, use the AddSeries function.

Examples

This statement adds a category named PCs to the graph gr_product_data:

```
gr_product_data.AddCategory("PCs")
```

See also

AddData
AddSeries
DeleteData
DeleteSeries

AddColumn

Description

Adds a column with a specified label, alignment, and width.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

ListView controls

Syntax

listviewname.**AddColumn** (*label, alignment, width*)

Argument	Description
listviewname	The name of the ListView control to which you want to add a column.
label	A string whose value is the name of the column you are adding.
alignment	A value of the enumerated datatype Alignment specifying the alignment of the column you are adding. Values are: • Center! • Justify! • Left! • Right!
width	An integer whose value is the width of the column you are adding, in PowerBuilder units.

Return value

Integer. Returns the column index if it succeeds and -1 if an error occurs.

Usage

The AddColumn function adds a column at the end of the existing columns unlike the InsertColumn function which inserts a column at a specified location.

Use SetItem and SetColumn to change the values for existing items. To add new items, use AddItem. To create columns for the report view of a ListView control, use AddColumn.

Examples

This script for a ListView event creates three columns in a ListView control:

```
integer index

FOR index = 3 to 25
    This.AddItem ("Category " + String (index), 1 )
NEXT

This.AddColumn("Name" , Left! , 1000)
This.AddColumn("Size" , Left! , 400)
This.AddColumn("Date" , Left! , 300)
```

See also

AddItem
DeleteColumn
InsertColumn

AddData

Adds a value to the end of a series of a graph. The syntax you use depends on the type of graph.

To add data to	Use
Any graph type except scatter	Syntax 1
Scatter graphs	Syntax 2

Syntax 1 For all graph types except scatter

Description

Adds a data point to a series in a graph. Use Syntax 1 for any graph type except scatter graphs.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Graph controls in windows and user objects. Does not apply to graphs within DataWindow objects because their data comes directly from the DataWindow.

Syntax

controlname.**AddData** (*seriesnumber*, *datavalue* {, *categoryvalue* })

Argument	Description
controlname	The name of the graph in which you want to add data to a series. The graph's type should not be scatter.
seriesnumber	The number that identifies the series to which you want to add data.
datavalue	The value of the data you want to add.
categoryvalue (optional)	The category for this data value on the category axis. The datatype of the *categoryvalue* should match the datatype of the category axis. In most cases you should include *categoryvalue*. Otherwise, an uncategorized value will be added to the series.

Return value

Long. Returns the position of the data value in the series if it succeeds and -1 if an error occurs. If any argument's value is null, AddData returns null.

Usage

When you use Syntax 1, AddData adds a value to the end of the specified series or to the specified category, if it already exists. If *categoryvalue* is a new category, the category is added to the end of the series with a label for the data point's tick mark. If the axis is sorted, the new category is incorporated into the existing order. If the category already exists, the new data replaces the old data at the data point for the category.

For example, if the third category label specified in series 1 is March and you add data in series 4 and specify the category label March, the data is added at data point 3 in series 4.

When the axis datatype is string, you can specify the empty string ("") as the category name. Because category names must be unique, there can be only one category with a blank name. If you use AddData to add data without specifying a category, you will have data points without categories, which is not the same as a category whose name is "".

To insert data in the middle of a series, use InsertData. You can also use InsertData to add data to the end of a series, as AddData does, although it requires an additional argument to do it.

For a comparison of AddData, InsertData, and ModifyData, see Equivalent Syntax in InsertData.

Examples

These statements add a data value of 1250 to the series named Costs and assign the data point the category label Jan in the graph gr_product_data:

```
integer SeriesNbr

// Get the number of the series.
SeriesNbr = gr_product_data.FindSeries("Costs")
gr_product_data.AddData(SeriesNbr, 1250, "Jan")
```

These statements add a data value of 1250 to the end of the series named Costs in the graph gr_product_data but do not assign the data point to a category:

```
integer SeriesNbr

// Get the number of the series.
SeriesNbr = gr_product_data.FindSeries("Costs")
gr_product_data.AddData(SeriesNbr, 1250)
```

See also

DeleteData
FindSeries
GetData
InsertData

Syntax 2 For scatter graphs

Description

Adds a data point to a series in a scatter graph.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

controlname.**AddData** (*seriesnumber*, *xvalue*, *yvalue)*

Argument	Description
controlname	The name of the scatter graph in which you want to add data to a series. The graph's type should be scatter.
seriesnumber	The number that identifies the series to which you want to add data.
xvalue	The x value of the data point you want to add.
yvalue	The y value of the data point you want to add.

Return value

Long. Returns the position of the data value in the series if it succeeds and -1 if an error occurs. If any argument's value is null, AddData returns null.

Examples

These statements add the x and y values of a data point to the series named Costs in the scatter graph gr_sales_yr:

```
integer SeriesNbr

// Get the number of the series.
SeriesNbr = gr_sales_yr.FindSeries("Costs")
gr_sales_yr.AddData(SeriesNbr, 12, 3)
```

See also

DeleteData
FindSeries
GetData

AddEntry

Description

Adds an entry to a dialing directory.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✗

Function not implemented
The AddEntry function is not implemented in PocketBuilder 2.0. It is reserved for future use.

Applies to

DialingDirectory objects

Syntax

objectname.AddEntry (*entry*)

Argument	Description
objectname	The name of the DialingDirectory object to which you want to add an entry.
entry	A DialingDirectoryEntry structure that you want to add to the directory. The DataSource property of the object must contain a positive non-zero value.

Return value

Integer. Returns 1 for success, and one of the following negative values if an error occurs:

-1	Unspecified error
-2	Not implemented error
-3	Supporting DLL not loaded
-4	Error in the passed-in arguments
-5	Initialization error

See also

AddRecipient
UpdateEntry

AddItem

Adds an item to a list control.

To add an item to	Use
A ListBox or DropDownListBox control	Syntax 1
A PictureListBox or DropDownPictureListBox control	Syntax 2
A ListView control when you only need to specify the item name and picture index	Syntax 3
A ListView control when you need to specify all the properties for the item	Syntax 4
A toolbar item to the Toolbar control	Syntax 5

Syntax 1 For ListBox and DropDownListBox controls

Description

Adds a new item to the list of values in a list box.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

ListBox and DropDownListBox controls

Syntax

listboxname.**AddItem** (*item*)

Argument	Description
listboxname	The name of the ListBox or DropDownListBox in which you want to add an item
item	A string whose value is the text of the item you want to add

Return value

Integer. Returns the position of the new item. If the list is sorted, the position returned is the position of the item after the list is sorted. Returns -1 if it fails. If any argument's value is null, AddItem returns null.

Usage

If the ListBox already contains items, AddItem adds the new item to the end of the list. If the list is sorted (its Sorted property is true), PocketBuilder re-sorts the list after the item is added.

A list can have duplicate items. Items in the list are tracked by their position in the list, not their text.

AddItem and InsertItem do not update the Items property array. You can use FindItem to find items added during execution.

Adding many items to a list with a horizontal scrollbar If a ListBox or the ListBox portion of a DropDownListBox will have a large number of items and you want to display an HScrollBar, call the SetRedraw function to turn Redraw off, add the items, call SetRedraw again to set Redraw on, and then set the HScrollBar property to true. Otherwise, it may take longer than expected to add the items.

Examples

This example adds the item Edit File to the ListBox lb_Actions:

```
integer rownbr
string s

s = "Edit File"
rownbr = lb_Actions.AddItem(s)
```

If lb_Actions contains Add and Run and the Sorted property is false, the statement above returns 3 (because Edit File becomes the third and last item). If the Sorted property is true, the statement above returns 2 (because Edit File becomes the second item after the list is sorted alphabetically).

See also

DeleteItem
FindItem
InsertItem
Reset
TotalItems

Syntax 2 For PictureListBox controls

Description

Adds a new item to the list of values in a picture list box.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

PictureListBox controls

Syntax

listboxname.**AddItem** (*item* {, *pictureindex* })

Argument	Description
listboxname	The name of the PictureListBox in which you want to add an item
item	A string whose value is the text of the item you want to add
pictureindex (optional)	An integer specifying the index of the picture you want to associate with the newly added item

Return value

Integer. Returns the position of the new item. If the list is sorted, the position returned is the position of the item after the list is sorted. Returns -1 if it fails. If any argument's value is null, AddItem returns null.

Usage

If you do not specify a picture index, the newly added item will not have a picture.

If you specify a picture index that does not exist, that number is still stored with the picture. If you add pictures to the picture array so that the index becomes valid, the item will then show the corresponding picture.

For additional notes about items in list boxes, see Syntax 1.

Examples

This example adds the item Cardinal to the PictureListBox plb_birds:

```
integer li_pic, li_position
```

```
string ls_name, ls_pic

li_pic = plb_birds.AddPicture("c:\pics\cardinal.bmp")
ls_name = "Cardinal"
li_position = plb_birds.AddItem(ls_name, li_pic)
```

If plb_birds contains Robin and Swallow and the Sorted property is false, the AddItem function above returns 3 because Cardinal becomes the third and last item. If the Sorted property is true, AddItem returns 1 because Cardinal is first when the list is sorted alphabetically.

See also

DeleteItem
FindItem
InsertItem
Reset
TotalItems

Syntax 3 For ListView controls

Description

Adds an item to a ListView control.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

ListView controls

Syntax

listviewname.**AddItem** (*label, pictureindex*)

Argument	Description
listviewname	The name of the ListView control to which you are adding a picture or item
label	The name of the item you are adding
pictureindex	The index of the picture you want to associate with the newly added item

Return value

Integer. Returns the index of the item if it succeeds and -1 if an error occurs.

Usage

Use this syntax if you only need to specify the label and picture index of the item you are adding to the ListView. If you need to specify more than the label and picture index, use Syntax 4.

Examples

This example uses AddItem in the Constructor event to add three items to a ListView control:

```
lv_1.AddItem("Sanyo" , 1)
lv_1.AddItem("Onkyo" , 1)
lv_1.AddItem("Aiwa" , 1)
```

See also

DeleteItem
FindItem
InsertItem
Reset
TotalItems

Syntax 4 For ListView controls

Description

Adds an item to a ListView control by referencing all the attributes in the ListView item.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

ListView controls

Syntax

listviewname.**AddItem** (*item*)

Argument	Description
listviewname	The name of the List View control to which you are adding a picture or item
item	The ListViewItem variable containing properties of the item you are adding

Return value

Integer. Returns the index of the item if it succeeds and -1 if an error occurs.

Usage

Use this syntax if you need to specify all the properties for the item you want to add. If you only need to specify the label and picture index, use Syntax 3.

Examples

This example uses AddItem in a CommandButton Clicked event to add a ListView item for each click:

```
count = count + 1
listviewitem l_lvi
l_lvi.PictureIndex = 2
l_lvi.Label = "Item "+ string(count)
lv_1.AddItem(l_lvi)
```

See also

DeleteItem
FindItem
InsertItem
Reset
TotalItems

Syntax 5 For Toolbar controls

Description

Adds a toolbar item to the toolbar control.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✗

Applies to

Toolbar controls

Syntax

Integer *controlname*.AddItem (*item*)

Argument	Description
controlname	The name of the toolbar control
item	Object of type ToolbarItem that you want to add to the toolbar

Return value

Integer. Returns the index of the item that you add to the toolbar.

Examples

The following example adds two toolbar items to the next available positions in the toolbar:

```
Integer li_rtn
ToolbarItem myItem
myItem.ItemPictureIndex = 1
myItem.ItemStyle = stylecheck!
li_rtn = tlbr_mytoolbar.AddItem(myItem)
myItem.ItemPictureIndex = 2
myItem.ItemStyle = stylebutton!
li_rtn = tlbr_mytoolbar.AddItem(myItem)
```

See also

DeleteItem
InsertItem

AddLargePicture

Description

Adds a bitmap, icon, or cursor to the large image list.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

ListView controls

Syntax

listviewname.**AddLargePicture** (*picturename*)

Argument	Description
listviewname	The name of the ListView control to which you are adding a bitmap, icon, or cursor
picturename	The name of the bitmap, icon, or cursor you are adding to the large image list

Return value

Integer. Returns the picture index if it succeeds and -1 if an error occurs.

Usage

When you add a large picture to a ListView, it is given the next available picture index in the ListView. For example, if your ListView has two pictures, the next picture you add will be assigned picture index number 3.

Before you add large pictures, you can specify scaling for the pictures by setting the LargePictureWidth and LargePictureHeight properties. The dimensions in effect when you add the first picture determine the scaling for all pictures. Changing the property values after you add pictures has no effect.

If you do not specify values for LargePictureWidth and LargePictureHeight before you add pictures, the dimensions of the first image determine the scaling for all pictures you add.

When you add a bitmap, specify the color in the bitmap that will be transparent by setting the LargePictureMaskColor property before calling AddLargePicture. You can change the LargePictureMaskColor property between calls.

Examples

This example adds the file *folder.ico*"to the large picture index of the ListView lv_files:

```
// Add large picture
integer index
index = lv_files.AddLargePicture("folder.ico")
```

See also

DeleteLargePicture

AddPicture

Adds a bitmap, icon, or cursor to a control.

To add a picture to	Use
A PictureListBox or TreeView control	Syntax 1
A Toolbar control	Syntax 2

Syntax 1 For PictureListBox and TreeView controls

Description

Adds a bitmap, icon, or cursor to the main image list.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

PictureListBox and TreeView controls

Syntax

controlname.**AddPicture** (*picturename*)

Argument	Description
controlname	The name of the control to which you want to add an icon, cursor, or bitmap to the main image list
picturename	The name of the icon, cursor, or bitmap you want to add to the main image list

Return value

Integer. Returns the picture index number if it succeeds and -1 if an error occurs.

Usage

The picture is assigned an index in the order in which it is added to the control.

Adding pictures during execution does not update the PictureName property array. Because the picture is added at the end of the list, the return value from AddPicture is the number of pictures associated with the control.

Before you add pictures, you can specify scaling for the pictures by setting the PictureWidth and PictureHeight properties. The dimensions in effect when you add the first picture determine the scaling for all pictures. Changing the property values after you add pictures has no effect.

If you do not specify values for PictureWidth and PictureHeight before you add pictures, the dimensions of the first image determine the scaling for all pictures you add.

When a you add a bitmap, specify the color in the bitmap that will be transparent by setting the PictureMaskColor property before calling AddPicture. You can change the PictureMaskColor property between calls.

Examples

This example adds a picture to a TreeView control and associates it with a new TreeView item:

```
long ll_tvi
integer li_picture
li_picture = &
tv_list.AddPicture("c:\apps_pb\staff.ico")
ll_tvi = tv_list.FindItem(RootTreeItem!, 0)
tv_list.InsertItemFirst(ll_tvi, "Dept.", li_picture)
```

See also

DeletePicture

Syntax 2 For Toolbar controls

Description

Adds a picture to the array of pictures available to the Toolbar control.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✗

Applies to

Toolbar controls

Syntax

Integer *controlname*.AddPicture (*picturename*)

Argument	Description
controlname	The name of the toolbar control
picturename	String for the name of the picture that you want to add for use by the toolbar

Return value

Integer. Returns 1 for success and -1 if an error occurs.

Examples

The following example adds a picture to the array of pictures available to be matched with items in the toolbar:

```
Integer li_rtn
li_rtn = tlbr_myToolBar.AddPicture &
 ("\program files\pic1.bmp")
```

See also

GetItemPictureIndex
SetItemPictureIndex

AddRecipient

Description

Adds the specified recipient for an appointment.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✗

Applies to

POOMAppointment controls

Syntax

Integer *objectname*.AddRecipient (*name* { *emailAddress* })

Argument	Description
objectname	The name of the POOMAppointment object
name	A string or an object of type POOMRecipient that specifies the name of a recipient to be added to the appointment's recipient list. If you use a POOMRecipient object for *name*, you cannot use the *emailAddress* argument. If you pass a string for *name*, but do not specify the *emailaddress*, or if you use a POOMRecipient object that does not contain an e-mail address, the Contacts list is searched for a matching name. If multiple matches are found, a Pocket Outlook dialog box displays so that a specific recipient can be selected.
emailAddress	(Optional) A string specifying the recipient's e-mail address.

Return value

Integer. Returns 1 for success and one of the following negative values if an error occurs:

-1	Unspecified error
-2	Cannot connect to the repository or a required internal subobject failed to connect to the repository
-3	Cannot log in to the repository
-4	Incorrect input argument
-5	Action cannot be performed
-6	The object identifier (OID) is not in the repository
-7	Feature is not implemented yet
-8	No matching entries found for the criteria

See also

GetRecipients
RemoveRecipient

AddSeries

Description

Adds a series to a graph, naming it with the specified name. The new series is also assigned a number. A graph's series are numbered consecutively, according to the order in which they are added.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Graph controls in windows and user objects. Does not apply to graphs within DataWindow objects because their data comes directly from the DataWindow.

Syntax

controlname.**AddSeries** (*seriesname*)

Argument	Description
controlname	The name of the graph in which you want to add a series
seriesname	A string whose value is the name of the series you want to add to *controlname*

Return value

Integer. Returns the number assigned to the series if it succeeds. If *seriesname* is a duplicate, AddSeries returns the number of the existing series. If an error occurs, it returns -1. If any argument's value is null, AddSeries returns null.

Usage

Adds *seriesname* to the graph *controlname* and assigns the series a number. The number identifies the series within the graph. The numbers are assigned in sequence. The first series you add to the graph is assigned number 1 and is the first series displayed in the graph; the next is assigned 2; and so on.

The series name must be unique within the graph. If you specify a name that already exists in the graph, AddSeries returns the number of the existing series. Series names are unique if they have different capitalization. The series name can be an empty string (""). However, because series names must be unique, only one series can have a blank name.

If you want to insert a series in the middle of the list, use InsertSeries. You can also use InsertSeries to add a series to the end of the list, as AddSeries does, although it requires an additional argument to do it.

To add data to a series in the graph, use the AddData or InsertData function. To add a category to a series, use the InsertCategory or AddCategory function.

Examples

These statements add the series named Costs to the graph gr_product_data:

```
integer series_nbr
series_nbr = gr_product_data.AddSeries("Costs")
```

These statements add an unnamed series to the graph gr_product_data:

```
integer series_nbr
series_nbr = gr_product_data.AddSeries("")
```

See also

AddCategory
AddData
DeleteData
DeleteSeries
FindSeries
InsertCategory
InsertSeries
SeriesCount
SeriesName

AddSmallPicture

Description

Adds a bitmap, icon, or cursor to the small image list.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

ListView controls

Syntax

listviewname.**AddSmallPicture** (*picturename*)

Argument	Description
listviewname	The name of the ListView control to which you are adding a small image
picturename	The name of the bitmap, icon, or cursor you are adding to the ListView control small image list

Return value

Integer. Returns the picture index if it succeeds and -1 if an error occurs.

Usage

When you add a small picture to a ListView control, it is given the next available picture index in the ListView. For example, if your ListView has two pictures, the next picture you add will have index number 3.

Before you add small pictures, you can specify scaling for the pictures by setting the SmallPictureWidth and SmallPictureHeight properties. The dimensions in effect when you add the first picture determine the scaling for all pictures. Changing the property values after you add pictures has no effect.

If you do not specify values for SmallPictureWidth and SmallPictureHeight before you add pictures, the dimensions of the first image determine the scaling for all pictures you add.

Before you call AddSmallPicture, specify the color in the bitmap that will be transparent by setting the SmallPictureMaskColor property. You can change the SmallPictureMaskColor property between calls.

Examples

This example adds the file "shortcut.ico" to the small picture index of the ListView lv_files:

```
//Add small picture
integer index
index = lv_files.AddSmallPicture("shortcut.ico")
```

See also

DeleteSmallPicture

AddStatePicture

Description

Adds a bitmap, icon, or cursor to the state image list.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

ListView and TreeView controls

Syntax

controlname.**AddStatePicture** (*picturename*)

Argument	Description
controlname	The name of the ListView or TreeView control to which you are adding a bitmap, cursor, or icon
picturename	The name of the bitmap, icon, or cursor you are adding

Return value

Integer. Returns the picture index if it succeeds and -1 if an error occurs.

Usage

For ListViews in large icon view, the state picture is a picture displayed to the left of the large picture, by default in a smaller size. For TreeViews, the state picture is displayed to the left of the regular picture and the item is moved to the right to make room for it.

If you specify either StatePictureWidth or StatePictureHeight, the picture is scaled to the size specified by that property.

When a you add a bitmap, specify the color in the bitmap that will be transparent by setting the StatePictureMaskColor property before calling AddPicture. You can change the StatePictureMaskColor property between calls.

Examples

This example adds the file *star.ico* to the state picture index of the ListView lv_files:

```
//Add state picture
integer index
index = lv_files.AddStatePicture("star.ico")
```

See also

DeleteStatePicture

AddToInfraredQueue

Description

Adds an appointment, contact, or task to the infrared queue.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✗

Applies to

POOM objects

Syntax

Integer *objectname*.AddToInfraredQueue (*entity*)

Argument	Description
objectname	The name of the POOM object
entity	Entity of type POOMAppointment, POOMContact, or POOMTask

Return value

Integer. Returns 1 for success and one of the following negative values if an error occurs:

-1 Unspecified error

-2 Cannot connect to the repository or a required internal subobject failed to connect to the repository

-3 Cannot log in to the repository

-4 Incorrect input argument

-5 Action cannot be performed

-6 The object identifier (OID) is not in the repository

-7	Feature is not implemented yet
-8	No matching entries found for the criteria

Usage

A user must be logged in to a POOM object to add an appointment, contact, or task to the infrared queue.

Examples

The following example submits the first appointment retrieved from Outlook to the infrared queue:

```
// Global variable: g_poom
Int li_rtn
POOMAppointment  myAppts[]
...
g_poom = CREATE POOM
li_rtn = g_poom.login()
// ** Gets and submits an appointment to the queue**
li_rtn = g_poom.getAppointments( myAppts )
li_rtn = g_poom.AddToInfraredQueue( myAppts[1])

g_poom.logout()
```

See also

Add
GetAppointment
GetContact
GetTask
ReceiveFromInfrared
SendToInfrared

AddToLibraryList

Description

Adds new files to the library search path of an application or component at runtime.

PocketBuilder	✗
PowerBuilder	✓

Syntax

AddToLibraryList (*filelist*)

Return value

Integer. Returns 1 if it succeeds. If an error occurs, it returns a negative value.

AllowReceivingCalls

Description

Allows reception of incoming calls.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✗

Applies to

PhoneCall objects

Syntax

objectname.AllowReceivingCalls (*allow*)

Argument	Description
objectname	The name of the PhoneCall object that will be allowed to receive calls
allow	A boolean indicating whether incoming calls will be accepted

Return value

Integer. Returns 1 for success and a negative value if an error occurs.

Usage

The AllowReceivingCalls function is typically called immediately after a PhoneCall object has been initialized.

Examples

In the following script for the cb_allow button, if the call has been initialized, the AllowReceivingCalls function is called and the cb_allow button’s text is toggled between “Disable Receive” and “Enable Receive”. The *g_phInit* global variable is set to 1 in the pcall_1 object’s constructor:

```
// Global variables: Long g_phInit = 0
// boolean gb_Allow
integer li_ret
if ( g_phInit > 0) then
   li_ret = pcall_1.AllowReceivingCalls(gb_Allow)
   if ( gb_Allow = true) then
      this.text = "Disable Receive"
      gb_Allow = false
   else
      this.text = "Enable Receive"
      gb_Allow = true
   end if
else
   sle_1.text = "Call not initialized"
end if
```

See also

AcceptCall
GetEntries
MakeCall
SetHold
SetMute
SetRingTone

Arrange

Description

Arranges the icons in rows.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

ListView controls

Syntax

listviewname.**Arrange** ( )

Argument	Description
listviewname	The name of the ListView control in which you want to arrange icons

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs.

Usage

Can only be used with large icon and small icon views.

Examples

This example arranges the icons in a ListView control:

```
lv_list.Arrange()
```

ArrangeSheets

Description

Arranges the windows contained in an MDI frame. (Windows that are contained in an MDI frame are called sheets.) You can arrange the open sheets and the icons of minimized sheets or just the icons.

PocketBuilder	✗
PowerBuilder	✓

Applies to
MDI frame windows

Syntax
mdiframe.**ArrangeSheets** (*arrangetype*)

Return value
Integer. Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is null, ArrangeSheets returns null.

Asc

Description
Converts the first character of a string to its ASCII integer value.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax
Asc (*string*)

Argument	Description
string	The string for which you want the ASCII value of the first character

Return value
Integer. Returns the ASCII value of the first character in *string*. If *string* is null, Asc returns null.

Usage
You can use Asc to find out the case of a character by testing whether its ASCII value is within the appropriate range.

Examples
This statement returns 65, the ASCII value for uppercase A:

```
Asc("A")
```

This example checks if the first character of string ls_name is uppercase:

```
String ls_name
IF Asc(ls_name) > 64 and Asc(ls_name) < 91 THEN ...
```

This example is a function that converts an array of integers into a string. Each integer specifies two characters. Its low byte is the first character in the pair and the high byte (ASCII * 256) is the second character. The function has an argument (iarr) which is the integer array:

```
string str_from_int, hold_str
integer arraylen

arraylen = UpperBound(iarr)

FOR i = 1 to arraylen
```

```
        // Convert first character of pair to a char
        hold_str = Char(iarr[i])

        // Add characters to string after converting
        // the integer's high byte to char
        str_from_int = &
        str_from_int + hold_str + &
        Char((iarr[i] - Asc(hold_str)) / 256)
    NEXT
```

For sample code that builds the integer array from a string, see Mid.

See also

Char
Mid
Asc method for DataWindows in the *DataWindow Reference* or online Help

ASin

Description

Calculates the arcsine of an angle.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

ASin (*n*)

Argument	Description
n	The ratio of the lengths of two sides of a triangle for which you want a corresponding angle (in radians). The ratio must be a value between -1 and 1.

Return value

Double. Returns the arcsine of *n*.

Examples

This statement returns .999998 (rounded to six places):

```
ASin(.84147)
```

This statement returns .520311 (rounded to six places):

```
ASin(LogTen (Pi (1)))
```

This statement returns 0:

```
ASin(0)
```

This code in the Clicked event of a button catches a runtime error that occurs when an arcsine is taken for a user-entered value—passed in a variable—that is outside of the permitted range:

```
Double ld_num
ld_num = Double (sle_1.text)

TRY
sle_2.text = string (asin (ld_num))
CATCH (runtimeerror er)
   MessageBox("Runtime Error", er.getmessage())
END TRY
```

See also

Sin
ACos
ATan
Pi
ASin method for DataWindows in the *DataWindow Reference* or online Help

ATan

Description

Calculates the arctangent of an angle.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

ATan (*n*)

Argument	Description
n	The ratio of the lengths of two sides of a triangle for which you want a corresponding angle (in radians)

Return value

Double. Returns the arctangent of *n*.

Examples

This statement returns 0:

```
ATan(0)
```

This statement returns 1.000 (rounded to three places):

```
ATan(1.55741)
```

This statement returns 1.267267 (rounded to six places):

```
ATan(Pi(1))
```

See also

Tan
ASin
ACos
ATan method for DataWindows in the *DataWindow Reference* or online Help

Beep

Description

Causes the computer to beep up to 10 times.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

Beep (*n*)

Argument	Description
n	The number of times you want the computer to beep. If *n* is greater than 10, the computer beeps 10 times.

Return value

Integer. Returns 1 if it succeeds and -1 if it fails. If *n* is null, Beep returns null. The return value usually is not used.

Examples

This statement causes the computer to beep five times:

```
Beep(5)
```

BeginPreview

Description

Starts the camera's preview mode. For HTC cameras, BeginPreview starts the IA Camera Wizard.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✗

Applies to

Camera objects

Syntax

objectname.BeginPreview (*previewwindow*)

Argument	Description
objectname	The name of the camera object that you want to inquire about
previewwindow	A graphic object such as a window or picture control that serves as the preview window

Return value

Integer. Returns 1 for success, and one of the following negative values if an error occurs:

-1	Unspecified error
-2	Supporting DLL not loaded
-3	Other initialization error
-5	Inconsistency in this object instance
-6	Call to the driver or device failed
-7	Unsupported option, such as setting a property the camera does not recognize
-8	Value for option is out of range
-9	Bad OS version (for example, expecting Windows Mobile 5 support on a Windows Mobile 2003 device)
-10	User pressed Cancel

Usage

The graphic control specified as an argument to the BeginPreview function is used only if the device supports preview in a specified control. Some devices preview directly to the physical screen.

Examples

The following example specifies that the image is previewed in the *p_preview* picture control on the window *w_main*:

```
li_rtn = g_myCamera.BeginPreview(w_main.p_preview)
```

See also

EndPreview
SetPreviewImageAttributes

BeginTransaction

Description Creates an EAServer transaction and associates it with the calling thread.

PocketBuilder	✗
PowerBuilder	✓

Applies to CORBACurrent objects

Syntax *CORBACurrent*.**BeginTransaction** ( )

Return value Boolean. Returns true if it succeeds and false if the transaction could not be created.

BitwiseAND

Description Allows you to compare the binary values of two variables with unsigned long datatypes.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✗

Syntax **BitwiseAND** (*val1*, *val2*)

Argument	Description
val1	Unsigned long for the first operand you want to compare
val2	Unsigned long for the second operand you want to compare

Return value UnsignedLong. Returns the matching bit or bits, or returns 0 when there is no match. If one of the arguments is null, BitwiseAND returns null.

Usage The bitwise AND operator compares each bit of its first operand to the corresponding bit of its second operand.

Examples This example compares the binary values of the *ll_a1* and *ll_a2* variables that are set to 10 (hexadecimal value 0x0A) and 12 (hexadecimal value 0x0C), respectively:

```
ULong ll_a1, ll_a2, ll_result
ll_a1 = Long ("0x0A")
ll_a2 = Long ("0x0C")
ll_result = BitwiseAND(ll_a1, ll_a2)
```

```
messagebox ("Result", string (ll_result, "hex4"))
```

In the above example, the first hexadecimal value has binary bits in the second and fourth positions. If you compare this to the second hexadecimal value, which has binary bits in the third and fourth positions, the only match is in the fourth position, which is equivalent to a decimal value of 8, or a four-place hexadecimal value of 0008.

See also

BitwiseNOT
BitwiseOR
BitwiseXOR

BitwiseClearBit

Description

Allows you to clear the bit from a specified position in a binary number.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✗

Syntax

BitwiseClearBit (*val*, *bit_Position*)

Argument	Description
val	Unsigned long for the binary number with a bit that you want to change from 1 to 0
bit_Position	Unsigned integer for the position, from 1 to 32, of the bit you want to clear

Return value

UnsignedLong. Returns the result for a binary number when a bit at the position specified by *bit_Position* is turned off. If one of its arguments is null, BitwiseClearBit returns null.

Usage

In PocketBuilder there are 32 bits in an unsigned long. If the value of *bit_Position* is 0, or greater than 32, PocketBuilder fires the runtime SystemError event because the value is out of range.

Examples

This example the variable *testValue* with its third bit cleared:

```
UnsignedLong ll_result
ll_result = BitwiseClearBit(testValue, 3)
```

In this example, if *testValue* is 12, it has bits in the third and fourth positions. The call to BitwiseClearBit turns off the bit in the third position and returns a value of 8. If *testValue* is 10, the bit in the third position is already turned off, so the value returned is 10.

See also

BitwiseGetBit
BitwiseSetBit
BitwiseShiftLeft
BitwiseShiftRight

BitwiseGetBit

Description

Allows you check whether a bit is turned on at a specified position of a binary number.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✗

Syntax

BitwiseGetBit (*val*, *bit_Position*)

Argument	Description
val	Unsigned long for the binary number with a bit that you want to check
bit_Position	Unsigned integer for the position, from 1 to 32, of the bit you want to check

Return value

Boolean. Returns true if the bit at the specified position is turned on and returns false if it is not. If one of its arguments is null, BitwiseGetBit returns null.

Usage

In PocketBuilder there are 32 bits in an unsigned long. If the value of *bit_Position* is 0, or greater than 32, PocketBuilder fires the runtime SystemError event because the value is out of range.

Examples

This example determines whether the values of the bit at the third position is on or off:

```
boolean lb_result
lb_result = BitwiseGetBit(testValue, 3)
```

In this example, if *testValue* is 12, it has bits in the third and fourth positions, and the call to BitwiseGetBit returns true. If *testValue* is 10, it has bits at the second and fourth positions only, so BitwiseGetBit returns false.

See also

BitwiseClearBit
BitwiseSetBit
BitwiseShiftLeft
BitwiseShiftRight

BitwiseNOT

Description

Allows you to get the bitwise complement of a binary number.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✗

Syntax

BitwiseNOT (*val1*)

Argument	Description
val1	Unsigned long for the binary number for which you want to get the bitwise complement

Return value

UnsignedLong. Returns the result of a bitwise complement for the *val1* binary value. If *val1* is null, BitwiseNOT returns null.

Usage

The BitwiseNOT function produces the bitwise complement of its sole operand. For each bit of the operand with a value of 1, the corresponding result bit is set to 0, and for each bit with a value of 0, the corresponding result bit is set to 1. BitwiseNOT resets the bits in all 32 positions for an unsigned long value in PocketBuilder.

Use hexadecimal formatting for capturing the return value.

Examples

This example returns the bitwise complement of the binary value of the *testValue* variable:

```
UnsignedLong ll_result
ll_result = BitwiseNOT(testValue)
```

In this example, if the *testValue* variable has a hexadecimal value of 0x10 (or a decimal value of 16), it has a bit set in its fifth position only. For this value, BitwiseNOT returns a binary number with set bits in all but the fifth position. This corresponds to a hexadecimal value of 0xffffffef , or a decimal value 4,294,967,279.

See also

BitwiseAND
BitwiseOR
BitwiseXOR

BitwiseOR

Description

Allows you to combine the bits from two binary numbers in an inclusive OR operation.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✗

Syntax

BitwiseOR (*val1*, *val2*)

Argument	Description
val1	Unsigned long for the first binary number that you want to use in an inclusive OR operation
val2	Unsigned long for the second binary number that you want to use in an inclusive OR operation

Return value

UnsignedLong. Returns the result of a bitwise OR operation. If one of its arguments is null, BitwiseOR returns null.

Usage

The bitwise “inclusive” OR operator compares each bit of its first operand to the corresponding bit of its second operand. If either or both operands has a bit value of 1 at a particular position, the result bit is set to 1 at that position. Otherwise, the corresponding result bit is set to 0.

Examples

This example compares the values of the bits in the *firstValue* and *secondValue* variables:

```
UnsignedLong ll_result
ll_result = BitwiseOR(firstValue, secondValue)
```

In this example, if *firstValue* is 10, it has positive bits in the second and fourth positions. If *secondValue* is 12, it has positive bits in the third and fourth positions. When you combine the binary values of 10 and 12 in a bitwise inclusive OR operation, the result is 14, with positive bits in the second, third, and fourth positions.

See also

BitwiseAND
BitwiseNOT
BitwiseXOR

BitwiseSetBit

Description

Allows you to set the bit at a specified position of a binary number.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✗

Syntax

BitwiseSetBit (*val1*, *bit_position*)

Argument	Description
val	Unsigned long for the binary number in which you want to set a particular bit
bit_position	Unsigned integer for the bit, from 1 to 32, that you want to set

Return value

UnsignedLong. Returns the result for a binary number when a bit at the position specified by *bit_Position* is turned on. If one of its arguments is null, BitwiseSetBit returns null.

Usage

In PocketBuilder there are 32 bits in an unsigned long. If the value of *bit_Position* is 0, or greater than 32, PocketBuilder fires the runtime SystemError event because the value is out of range.

Examples

This example sets the bit in the third position of the *testValue* variable.

```
UnsignedLong ll_result
ll_result = BitwiseSetBit(testValue, 3)
```

In this example, if *testValue* is 10, with bits in the second and fourth positions, the return value is a binary number with bits set at the second, third, and fourth positions. This converts to a decimal value of 14.

If *testValue* is 12, it has bits set in the third and fourth positions. In this case, calling BitwiseSetBit with a *bit_position* value of 3 does not change the original value, since it already has a set bit at its third position.

See also

BitwiseClearBit
BitwiseGetBit
BitwiseShiftLeft
BitwiseShiftRight

BitwiseShiftLeft

Description

Allows you to modify a binary number by shifting all its bits to the left by a specified amount.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✗

Syntax

BitwiseShiftLeft (*val*, *shift_amount*)

Argument	Description
val	Unsigned long for the binary number you want to modify
shift_amount	Unsigned integer for the number of places you want to shift the bits of a binary number

Return value

UnsignedLong. Returns the result for a binary number when its bits are shifted to the left by the amount specified in the *shift_amount* argument. If one of its arguments is null, BitwiseShiftLeft returns null.

Usage

In PocketBuilder there are 32 bits in an unsigned long. If you enter 31 or greater for *shift_amount*, PocketBuilder fires the runtime SystemError event because the value is out of range.

Examples

This example shifts the bits in *testValue* two places to the left:

```
UnsignedLong ll_result
ll_result = BitwiseShiftLeft(testValue, 2)
```

In this example, when *testValue* is 10, its binary equivalent has positive bits in the second and fourth positions. If you shift the bits two places to the left, the resulting number is 40, with positive bits in the fourth and sixth positions.

See also

BitwiseClearBit
BitwiseGetBit
BitwiseSetBit
BitwiseShiftRight

BitwiseShiftRight

Description

Allows you to modify a binary number by shifting all its bits to the right by a specified amount.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✗

Syntax

BitwiseShiftRight (*val*, *shift_amount*)

Argument	Description
val	Unsigned long for the binary number you want to modify
shift_amount	Unsigned integer for the number of places you want to shift the bits of a binary number

Return value

UnsignedLong. Returns the result for a binary number when its bits are shifted to the right by the amount specified in the *shift_amount* argument. If one of its arguments is null, BitwiseShiftRight returns null.

Usage

In PocketBuiilder there are 32 bits in an unsigned long. If you enter 31 or greater for *shift_amount*, PocketBuilder fires the runtime SystemError event because the value is out of range.

Examples

This example shifts the bits in *testValue* three places to the right:

```
UnsignedLong ll_result
ll_result = BitwiseShiftRight(testValue, 2)
```

In this example, when *testValue* is 10, its binary equivalent has bits in the second and fourth positions. If you shift the bits two places to the right, the resulting number is 2, with a bit set in the second position only. The other bit was moved out of range.

See also

BitwiseClearBit
BitwiseGetBit
BitwiseSetBit

BitwiseShiftLeft

BitwiseXOR

Description

Allows you to compare the bits from two binary numbers, setting all matched bits to 0 and mismatched bits to 1.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✗

Syntax

BitwiseXOR (*val1*, *val2*)

Argument	Description
val1	Unsigned long for the first binary number you want to compare
val2	Unsigned long for the second binary number you want to compare

Return value

UnsignedLong. Returns the result of a bitwise XOR operation. If one of its arguments is null, BitwiseXOR returns null.

Usage

The bitwise "exclusive" OR operator compares each bit of its first operand to the corresponding bit of its second operand. If the bit at a particular position of one operand is 0 and the bit at the same position in the other operand is 1, the corresponding result bit is set to 1. Otherwise, the corresponding result bit is set to 0.

Examples

This example the values of the bits in the *firstValue* and *secondValue* variables:

```
UnsignedLong ll_result
ll_result = BitwiseXOR(firstValue, secondValue)
```

In this example, if *firstValue* is 10, it has bits in the second and fourth positions. If *secondValue* is 12, it has bits in the third and fourth positions. When you compare the binary values of 10 and 12 in a bitwise exclusive OR operation, the result is 6, with bits set in the second and third positions only.

See also

BitwiseAND
BitwiseNOT
BitwiseOR

Blob

Description

Converts a string to a blob datatype.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

Blob (*text*)

Argument	Description
text	The string you want to convert to a blob datatype

Return value

Blob. Returns the converted string. If *text* is null, Blob returns null.

Examples

This example saves a text string as a blob datatype:

```
Blob B
B = Blob("Any Text")
```

See also

BlobEdit
BlobMid
String

BlobEdit

Description

Inserts data of any PocketBuilder datatype into a blob variable.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

BlobEdit (*blobvariable*, *n*, *data*)

Argument	Description
blobvariable	An initialized variable of the blob datatype into which you want to copy a standard PocketBuilder datatype
n	The number (1 to 4,294,967,295) of the position in *blobvariable* at which you want to begin copying the data
data	Data of a valid PocketBuilder datatype that you want to copy into *blobvariable*

Return value

Unsigned long. Returns the position at which the next data can be copied if it succeeds, and returns null if there is not enough space in *blobvariable* to copy the data. If any argument's value is null, BlobEdit returns null.

If the *data* argument is a string, the position in the *blobvariable* in which you want to copy data will be the length of the string + 2. If the *data* argument is a string converted to a blob, the position will be the length of the string + 1. This is because a string contains a null terminating character that it loses when it is converted to a blob. Thus, `BlobEdit (blob_var, 1, "ZZZ'')` returns 5, while `BlobEdit (blob_var, 1, blob (''ZZZ'') )` returns 4.

Examples

This example copies a bitmap in the blob emp_photo starting at position 1, stores the position at which the next copy can begin in *nbr*, and then copies a date into the blob emp_photo after the bitmap data:

```
blob{1000} emp_photo
blob temp
date pic_date
ulong nbr

... // Read BMP file containing employee picture
... // into temp using FileOpen and FileRead.
pic_date = Today()

nbr = BlobEdit(emp_photo, 1, temp)
BlobEdit(emp_photo, nbr, pic_date)
UPDATEBLOB Employee SET pic = :emp_photo
    WHERE ...
```

See also

Blob
BlobMid

BlobMid

Description

Extracts data from a blob variable.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

BlobMid (*data*, *n* {, *length* })

Argument	Description
data	Data of the blob datatype
n	The number (1 to 4,294,967,295) of the first byte you want returned
length (optional)	The number of bytes (1 to 4,294,967,295) you want returned

Return value

Blob. Returns *length* bytes from *data* starting at byte *n*. If *n* is greater than the number of bytes in *data*, BlobMid returns an empty blob. If together *length* and *n* add up to more bytes than the blob contains, BlobMid returns the remaining bytes, and the returned blob will be shorter than the specified length. If any argument's value is null, BlobMid returns null.

Include terminator character
String variables contain a zero terminator, which accounts for one byte. Include the terminator character when calculating how much data to extract.

Examples

In this example, the first call to BlobMid stores 10 bytes of the blob *datablob* starting at position 5 in the blob *data_1*; the second call stores the bytes of datablob from position 5 to the end in *data_2*:

```
blob data_1, data_2, datablob

... // Read a blob datatype into datablob.

data_1 = BlobMid(datablob, 5, 10)
data_2 = BlobMid(datablob, 5)
```

This code copies a bitmap in the blob *emp_photo* starting at position 1, stores the position at which the next copy can begin in *nbr*, and then copies a date into the blob *emp_photo* after the bitmap data. Then, using the date's start position, it extracts the date from the blob and displays it in the StaticText st_1:

```
blob{1000} emp_photo
blob temp
date pic_date
ulong nbr

... // Read BMP file containing employee picture
... // into temp using FileOpen and FileRead.

pic_date = Today()
nbr = BlobEdit(emp_photo, 1, temp)
BlobEdit(emp_photo, nbr, pic_date)
st_1.Text = String(Date(BlobMid(emp_photo, nbr)))
```

See also	Blob BlobEdit

BuildModel

Description

Builds either a performance analysis or trace tree model based on the trace file you have specified with the SetTraceFileName function. Optional arguments let you monitor the progress of the build or interrupt it.

PocketBuilder on Desktop	✓
PocketBuilder on Pocket PC	✗
PocketBuilder on Smartphone	✗
PowerBuilder	✓

Applies to

Profiling and TraceTree objects

Syntax

instancename.**BuildModel** ({ *progressobject, eventname, triggerpercent* })

Argument	Description
instancename	Instance name of the Profiling or TraceTree object
progressobject (optional)	A PowerObject that represents the number of activities that have been processed
eventname (optional)	A string specifying the name of an event you define
triggerpercent (optional)	A long identifying the number of activities the BuildModel function should process before triggering the *eventname* event

Return value

ErrorReturn. Returns one of the following values:

- Success!—The function succeeded
- FileNotSetError!—TraceFileName has not been set
- ModelExistsError!—A model has already been built
- EnterpriseOnlyFeature!—This function is supported only in the Enterprise edition of PowerBuilder
- EventNotFoundError!—The event cannot be found on the passed *progressobject*, so the model cannot be built
- EventWrongPrototypeError!—The event was found but does not have the proper prototype, so the model cannot be built

- SourcePBLError!—The source libraries cannot be found, so the model cannot be built

Usage

You must specify the trace file to be modeled using the SetTraceFileName function before calling BuildModel.

The BuildModel function extracts raw data from a trace file and maps it to objects that can be acted upon by PowerScript functions. If you want to build a model of your trace file without recording the progress of the build, call BuildModel without any of its optional arguments. If you want to receive progress information while the model is being created or if you want to be able to interrupt a BuildModel that is taking too long to complete, call BuildModel with its optional arguments.

The event *eventname* on the passed *progressobject* is triggered when the number of activities indicated by the *triggerpercent* argument are processed. If the value of *triggerpercent* is 0, *eventname* is triggered for every activity. If the value of *triggerpercent* is greater than 100, *eventname* is never triggered. You define this event using this syntax:

eventname (*currentactivity, totalnumberofactivities*)

Argument	Description
eventname	Name of the event
currentactivity	A long identifying the number of the current activity
totalnumberofactivities	A long identifying the total number of activities in the trace file

Eventname returns a boolean value. If it returns false, the processing initiated by the BuildModel function is canceled and any temporary storage is cleaned up. If you need to stop BuildModel processing that is taking too long, you can return a false value from *eventname*. The script you write for *eventname* determines how progress is monitored. For example, you might display progress or simply check whether the processing must be canceled.

Examples

This example creates a performance analysis model of a trace file:

```
Profiling lpro_model
String ls_filename

lpro_model = CREATE Profiling
lpro_model.SetTraceFileName(ls_filename)
lpro_model.BuildModel()
```

This example creates a trace tree model of a trace file:

```
TraceTree ltct_model
String ls_filename
```

```
ltct_model = CREATE TraceTree
ltct_model.SetTraceFileName(ls_filename)
ltct_model.BuildModel()
```

This example creates a performance analysis model that provides progress information as the model is built. The *eventname* argument to BuildModel is called ue_progress and is triggered each time five percent of the activities have been processed. The progress of the build is shown in a window called w_progress that includes a Cancel button:

```
Profiling lpro_model
String ls_filename
Boolean lb_cancel

lpro_model = CREATE Profiling
lb_cancel = false
lpro_model.SetTraceFileName(ls_filename)

Open(w_progress)
// Call the of_init window function to initialize
// the w_progress window
w_progress.of_init(lpro_model.NumberOfActivities, &
    'Building Model', This, 'ue_cancel')

lpro_model.BuildModel(This, 'ue_progress', 5)

// Clicking the cancel button in w_progress
// sets lb_cancel to true and returns
// false to ue_progress
IF lb_cancel THEN &
    Close(w_progress)
    RETURN -1
END IF
```

See also

SetTraceFileName
DestroyModel

Cancel

Stops the execution of a pipeline or to send a cancellation notice to the recipient of a Pocket Outlook appointment.

To cancel	Use
Execution of a pipeline object	Syntax 1
An appointment for a POOMAppointment object	Syntax 2

Syntax 1 For pipeline objects

Description

Stops the execution of a pipeline object.

PocketBuilder	✗
PowerBuilder	✓

Applies to

Pipeline objects

Syntax

pipelineobject.**Cancel** ( )

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs.

Syntax 2 For POOMAppointment objects

Description

Sends a cancellation notice to the appointment's recipients, but does not remove the appointment from the repository. Call the Remove function on the POOM object to remove the appointment from the repository.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✗

Applies to

POOMAppointment objects

Syntax

Integer *objectname*.Cancel ( )

Argument	Description
objectname	The name of the POOMAppointment object

Return value

Integer. Returns 1 for success and one of the following negative values if an error occurs:

-1	Unspecified error
-2	Cannot connect to the repository or a required internal subobject failed to connect to the repository
-3	Cannot log in to the repository
-4	Incorrect input argument
-5	Action cannot be performed
-6	The object identifier (OID) is not in the repository
-7	Feature is not implemented yet
-8	No matching entries found for the criteria

See also

Remove
Update

CancelSync

Cancels a synchronization process.

To cancel	Use
System function	Syntax 1
Synchronization object function	Syntax 2

Syntax 1 System function

Description

Closes the window associated with a current synchronization process.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✗

Syntax

CancelSync (*windowclassname*)

Argument	Description
windowclassname	A read-only string for the handle to the window receiving synchronization messages

Return value	Long. Returns 1 for success and -1 for failure.
Usage	If you use the MobiLink Synchronization for ASA wizard, it is typically not necessary to call *CancelSync* explicitly. This is because the uf_cancelsync function of the wizard-generated user object makes an equivalent call to pb_cancel_dbmlsync in the PocketBuilder VM. (The uf_cancelsync function is called, for example, in the Clicked event of the Cancel button on the wizard-generated synchronization window.)
See also	RunSync

Syntax 2 Synchronization object function

Description: Reserved for future use. Cancels the synchronization process and rolls back any changes accumulated during the processing.

Applies to: MLSynchronization, MLSync controls

Syntax: *SyncObject*.**CancelSync** ( )

Argument	Description
syncObject	The name of the synchronization object that started a synchronization process that you want to stop.

Return value: Integer. Returns 1 for success and -1 for failure.

CanUndo

Description: Tests whether Undo can reverse the most recent edit for an editable control.

PocketBuilder	✗
PowerBuilder	✓

Applies to: Any editable control (DataWindow, MultiLineEdit, SingleLineEdit, RichTextEdit)

Syntax: *editname*.**CanUndo** ( )

Return value: Boolean. Returns true if the last edit can be reversed (undone) using the Undo function and false if the last edit cannot be reversed. If *editname* is null, CanUndo returns null.

CaptureImage

Description	Captures an image and saves it as a file.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✗

Applies to — Camera objects

Syntax — *objectname*.CaptureImage (*fileName*)

Argument	Description
objectname	The name of the camera object that you want to inquire about
fileName	A string that specifies the name of the file to which the image is to be saved

Return value — Integer. Returns 1 for success, and one of the following negative values if an error occurs:

-1	Unspecified error
-2	Supporting DLL not loaded
-3	Other initialization error
-5	Inconsistency in this object instance
-6	Call to the driver or device failed
-7	Unsupported option, such as setting a property the camera does not recognize
-8	Value for option is out of range
-9	Bad OS version (for example, expecting Windows Mobile 5 support on a Windows Mobile 2003 device)
-10	User pressed Cancel

Usage — The image is saved as a file in JPEG format, or in a format specified using the CamOptCaptureFormat! value of the CameraOption enumerated variable. Some devices stop the preview when the capture begins and restart preview when the capture is complete. For a list of values of the CameraOption enumerated variable, see GetOption.

HTC cameras
You cannot use PowerScript image-capturing functions with cameras that depend on the IA Camera Wizard for these functions. Instead, you can use the Camera object Snapped event to retrieve the file name for an image that you capture using the IA Camera Wizard.

Examples

The following example tests whether the image is ready to be captured before capturing it:

```
if g_mycamera.IsReadyForCapture() then
   li_rtn = g_myCamera.CaptureImage("\mypic.jpg")
end if
```

See also

BeginPreview
EndPreview
GetAllowedImageAttributes
IsReadyToCapture
Open
SetCaptureImageAttributes
SetPreviewImageAttributes

CategoryCount

Description

Counts the number of categories on the category axis of a graph.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Graph controls in windows and user objects, and graphs in DataWindow controls

Syntax

controlname.**CategoryCount** ({ *graphcontrol* })

Argument	Description
controlname	The name of the graph for which you want the number of categories, or the name of a DataWindow control containing the graph.

Argument	Description
graphcontrol (DataWindow control only)	A string whose value is the name of the graph in the DataWindow for which you want the number of categories. *Graphcontrol* is required only if *controlname* is a DataWindow control.

Return value

Integer. Returns the count if it succeeds and -1 if an error occurs. If any argument's value is null, CategoryCount returns null.

Examples

These statements get the number of categories in the graph gr_revenues in the DataWindow control dw_findata:

```
integer li_count
li_count = &
    dw_findata.CategoryCount("gr_revenues")
```

These statements get the number of categories in the graph gr_product_data:

```
integer li_count
li_count = gr_product_data.CategoryCount()
```

See also

DataCount
SeriesCount

CategoryName

Description

Obtains the category name associated with the specified category number.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Graph controls in windows and user objects, and graphs in DataWindow controls .

Syntax

controlname.**CategoryName** ({ *graphcontrol*, } *categorynumber*)

Argument	Description
controlname	The name of the graph in which you want to find the name of a specific category, or the name of the DataWindow control containing the graph.

Argument	Description
graphcontrol (DataWindow control only)	A string whose value is the name of the graph in the DataWindow for which you want the name of a specific category. *Graphcontrol* is required only if *controlname* is a DataWindow control.
categorynumber	The number of the category for which you want the name.

Return value

String. Returns the name of *categorynumber* in *controlname*. If an error occurs, it returns the empty string (""). If any argument's value is null, CategoryName returns null.

Usage

Categories are numbered consecutively, from 1 to the value returned by CategoryCount. When you delete a category, the categories are renumbered to keep the numbering consecutive. You can use CategoryName to find out the named category associated with a category number.

Examples

These statements obtain the name of category 5 in the graph gr_product_data:

```
string ls_name
ls_name = gr_product_data.CategoryName(5)
```

These statements obtain the name of category 5 in the graph gr_revenues in the DataWindow control dw_findata:

```
string ls_name
ls_name = &
    dw_findata.CategoryName("gr_revenues", 5)
```

See also

AddCategory
SeriesName

Ceiling

Description

Determines the smallest whole number that is greater than or equal to a specified limit.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

Ceiling (*n*)

Argument	Description
n	The number for which you want the smallest whole number that is greater than or equal to it

Return value

The datatype of *n*. Returns the smallest whole number that is greater than or equal to *n*. If *n* is null, Ceiling returns null.

Examples

These statements set num to 5:

```
decimal dec, num
dec = 4.8
num = Ceiling(dec)
```

These statements set num to –4:

```
decimal num
num = Ceiling(-4.2)
num = Ceiling(-4.8)
```

See also

Int
Round
Truncate
Ceiling method for DataWindows in the *DataWindow Reference* or online Help

ChangeDirectory

Description

Changes the current directory.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

ChangeDirectory (*directoryname*)

Argument	Description
directoryname	String for the name of the directory you want to set as the current directory

Return value

Integer. Returns 1 if the function succeeds and -1 if an error occurs.

Examples

This example changes the current directory to the parent directory of the current directory and displays the new current directory in a SingleLineEdit control:

```
ChangeDirectory( ".." )
sle_1.text= GetCurrentDirectory( )
```

See also

CreateDirectory
GetCurrentDirectory

ChangeMenu

Description

Changes the menu associated with a window.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Window objects

Syntax

windowname.**ChangeMenu** (*menuname* {, *position* })

Argument	Description
windowname	The name of the window for which you want to change the menu.
menuname	The name of the menu you want to make the current menu.
position (PowerBuilder only)	For an MDI frame window, the number of the item on the menu bar to which you want to append the names of the open sheets. Items on the menu bar are numbered from the left, beginning with 1. The default is 0, which lists the open sheets on the menu bar's next-to-last menu (or the last menu if there is only one available).

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is null, ChangeMenu returns null. The return value is usually not used.

Examples

This statement changes the top-level menu of the w_Employee window to m_Emp1:

```
w_Employee.ChangeMenu(m_Emp1)
```

Char

Description

Extracts the first character of a string or converts an integer to a char.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

Char (*n*)

Argument	Description
n	A string that begins with the character you want, an integer you want to convert to a character, or a blob in which the first value is a string or integer. The rest of the contents of the string or blob is ignored. *N* can also be an Any variable containing a string, integer, or blob.

Return value

Char. Returns the first character of *n*. If *n* is null, Char returns null.

Examples

This example sets *ls_S* to an asterisk, the character corresponding to the ASCII value 42:

```
string ls_S
ls_S = Char(42)
```

These statements generate delivery codes A to F for the values 1 through 6 of *li_DeliveryNbr*:

```
string ls_Delivery
integer li_DeliveryNbr

FOR li_DeliveryNbr = 1 to 6
    ls_Delivery = Char(64 + li_DeliveryNbr)
    ... // Additional processing of ls_Delivery
NEXT
```

See also

Asc
Char method for DataWindows in the *DataWindow Reference* or online Help

Check

Description

Displays a check mark next to a menu item in a drop-down or cascading menu and sets the menu item's Checked property to true.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✓

Applies to

Menu objects

Syntax

menuname.**Check** ( )

Argument	Description
menuname	The fully qualified name of the menu next to which you want to display a check mark. The item must be in a drop-down or cascading menu, not an item on a menu bar.

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs. If *menuname* is null, Check returns null.

Usage

A check mark next to a menu item indicates that the menu option is currently on and that the user can turn the option on and off by choosing it. For example, in the Window painter's Design menu, a check mark is displayed next to Grid when the grid is on.

You can use Check in an item's Clicked script to mark a menu item when the user turns the option on and Uncheck to remove the check when the user turns the option off.

Equivalent syntax You can set a menu object's Checked property instead of calling Check.

menuname.Checked = true

This statement:

```
Menu_Appl.M_View.M_Grid.Checked = TRUE
```

is equivalent to:

```
Menu_Appl.M_View.M_Grid.Check()
```

Examples

This statement displays a check mark next to the menu item m_Grid in the m_View drop-down menu on the menu bar m_Appl:

```
m_Appl.m_View.m_Grid.Check()
```

See also

Uncheck

ChooseColor

Description

Displays the standard color selection dialog box.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

ChooseColor (*color* {, *customcolors* [ ] })

Argument	Description
color	A long passed by reference that represents the color selected in the dialog box
customcolors (optional)	A long array of custom colors passed by reference to the color selection dialog box

Return value

Integer. Returns 1 if the function succeeds, 0 if the user selects cancel (or the dialog box is closed), -1 if an error occurs.

Examples

This example displays the color selection dialog box with a base color of red and with two different custom colors defined:

```
long  red, green, blue
long custom[ ]
integer li_color

red = 255
custom[1]=rgb(red, 200, blue)
custom[2]=8344736
li_color = ChooseColor( red, custom [ ] )
```

See also

RGB

ClassList

Description

Provides a list of the classes included in a performance analysis model.

PocketBuilder on Desktop	✓
PocketBuilder on Pocket PC	✗
PocketBuilder on Smartphone	✗
PowerBuilder	✓

Applies to

Profiling object

Syntax

instancename.**ClassList** (*list*)

Argument	Description
instancename	Instance name of the Profiling object.
list	An unbounded array variable of datatype ProfileClass in which ClassList stores a ProfileClass object for each class included in the model. This argument is passed by reference.

Return value

ErrorReturn. Returns one of the following values:

- Success!—The function succeeded
- ModelNotExistsError!—The function failed because no model exists

Usage

You use the ClassList function to extract a list of the classes included in a performance analysis model. You must have previously created the performance analysis model from a trace file using the BuildModel function. Each class listed is defined as a ProfileClass object and provides the class name, its parent class and type, and a list of the routines associated with that class. The classes are listed in no particular order.

Examples

This example lists the classes included in the performance analysis model:

```
ProfileClass lproclass_list[], lproclass_class
Profiling lpro_model
Long ll_limitclass, ll_indexclass

lpro_model = CREATE Profiling
lpro_model.BuildModel()

lpro_model.ClassList(lproclass_list)
ll_limitclass = UpperBound(lproclass_list)

FOR ll_indexclass = 1 TO ll_limitclass
    lproclass_class = lproclass_list[ll_indexclass]
    ...
NEXT
```

See also

BuildModel

ClassName

Determines the class of an object or the datatype of a variable.

To determine	Use
The class of an object	Syntax 1
The class (or datatype) of a variable	Syntax 2

Syntax 1 For any object

Description

Provides the class (or name) of the specified object.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Any control

Syntax

controlname.**Classname** ( )

Argument	Description
controlname	The name of the control for which you want to know the name assigned to the control in the style window (the class of the control)

Return value

String. Returns the class of *controlname*, the name assigned to the control. Returns the empty string ("") if an error occurs. If *controlname* is null, ClassName returns null.

Usage

The class is the name of an object. You assign the name when you save the object in its painter. Usually the class and the object itself appear to be the same (because PocketBuilder declares a variable with the same name as the class for the object). However, if you have declared multiple instances of an object, it is clear that the object's class and the object's variable are different.

If an ancestor object has been instantiated with one of its descendants, you can use ClassName to find the name of the descendant.

TypeOf reports an object's built-in object type. The types are values of the Object enumerated datatype, such as Window! or CheckBox!. ClassName reports the class of the object in the ancestor-descendant hierarchy.

Examples

These statements return the class of the dragged control *Source*:

```
DragObject Source
string which_class
```

```
Source = DraggedObject()
which_class = Source.ClassName()
```

These statements return the class of the objects in the control array and store them in *the_class* array:

```
string the_class[]
windowobject the_object[]
integer i

FOR i = 1 TO UpperBound(control[])
    the_object[i] = control[i]
    the_class[i] = the_object[i].ClassName()
NEXT
```

See also

DraggedObject
TypeOf

Syntax 2 For variables

Description

Provides the datatype of a variable.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

ClassName (*variable*)

Argument	Description
variable	The name of the variable for which you want to know its name (that is, its datatype)

Return value

String. Returns the name of *variable*. Returns the empty string ("") if *variable* is an enumerated datatype or if an error occurs. If *variable* is null, ClassName returns null.

Usage

ClassName cannot determine the datatype if *variable* is an enumerated datatype. In this case, ClassName returns the empty string.

Examples

If *gd_double* is a global double variable, ClassName sets *varname* to double:

```
string varname
varname = ClassName(gd_double)
```

Clear

Clears selected text or other contents of a specified control.

To clear	Use
Selected text from a specified control	Syntax 1
The contents of a Signature control	Syntax 2

Syntax 1 For edit and list box controls

Description

Deletes selected text from the specified control, but does not store it in the clipboard.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

DataWindow, EditMask, MultiLineEdit, SingleLineEdit, RichTextEdit, DropDownListBox, DropDownPictureListBox, OLE controls, and OLEStorage objects

Syntax

objectname.**Clear** ( )

Argument	Description
objectname	One of the following: • The name of the control from which you want to delete (clear) selected text. • The name of an OLE control or storage object variable (type OLEStorage) from which you want to release its OLE object. If *objectname* is a DropDownListBox its AllowEdit property must be true.

Return value

Long.

For edit controls, returns the number of characters that Clear removed from *objectname*. If no text is selected, no characters are removed and Clear returns 0. If an error occurs, Clear returns -1.

If *objectname* is null, Clear returns null.

Usage

To select text for deleting, the user can use the mouse or keyboard. You can also call the SelectText function in a script.

To delete selected text and store it in the clipboard, use the Cut function.

Examples

If the text in sle_comment1 is Draft and it is selected, this statement clears Draft from sle_comment1 and returns 5:

```
sle_comment1.Clear()
```

If the text in sle_comment1 is Draft, the first statement selects the D and the second clears D from sle_comment1 and returns 1:

```
sle_comment1.SelectText(1,1)
sle_comment1.Clear()
```

See also

Close
Cut
Paste
ReplaceText
SelectText

Syntax 2 For Signature controls

Description

Clears the contents of the control.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✗

Applies to

Signature controls

Syntax

void *controlname*.Clear ( )

Argument	Description
controlname	The name of the signature control you want to clear

Return value

None

Examples

This statement clears the contents of the signature control *sig_1*:

```
sig_1.clear()
```

ClearRecurrencePattern

Description

Clears the recurrence pattern for an appointment and sets it as an appointment with a single instance.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✗

Applies to

POOMAppointment, POOMTask objects

Syntax

Integer *objectname*.ClearRecurrencePattern ( )

Argument	Description
objectname	The name of the POOMAppointment or POOMTask object

Return value

Integer. Returns 1 for success and one of the following negative values if an error occurs:

-1	Unspecified error
-2	Cannot connect to the repository or a required internal subobject failed to connect to the repository
-3	Cannot log in to the repository
-4	Incorrect input argument
-5	Action cannot be performed
-6	The object identifier (OID) is not in the repository
-7	Feature is not implemented yet
-8	No matching entries found for the criteria

See also

GetRecurrence
SetRecurrence
SkipRecurrence

Clipboard

Retrieves or replaces the contents of the system clipboard.

To	Use
Retrieve or replace the contents of the system clipboard with text	Syntax 1
Replace the contents of the system clipboard with a bitmap image of a graph	Syntax 2

Syntax 1 For text

Description

Retrieves or replaces the contents of the system clipboard with text.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

Clipboard ({ *string* })

Argument	Description
string (optional)	A string whose value is the text you want to place in the clipboard. The string replaces the current contents of the clipboard, if any.

Return value

String. Returns the current contents of the clipboard if the clipboard contains text. If *string* is specified, Clipboard returns the current contents and replaces it with *string*.

Returns the empty string ("") if the clipboard is empty or it contains nontext data, such as a bitmap. If *string* is specified, the nontext data is replaced with *string*. If *string* is null, Clipboard returns null.

Usage

You can use Syntax 1 with the Paste, Replace, or ReplaceText function to insert the clipboard contents in an editable control or StaticText control.

Calling Clipboard in a DataWIndow control or DataStore object To retrieve or replace the contents of the system clipboard with text from a DataWindow item (cell value), you must first assign the value to a string and then call the system Clipboard function as follows:

```
string ls_data = dw_1.object.column_name[row_number]
::Clipboard(ls_data)
```

The DataWindow version of Clipboard, documented in Syntax 2 (and in the *DataWindow Reference*), is only applicable to graphs.

Examples

These statements put the contents of the clipboard in the variable *ls_CoName*:

```
string ls_CoName
ls_CoName = Clipboard()
```

The following statements place the contents of the clipboard in *Heading*, and then replace the contents of the clipboard with the string Employee Data:

```
string Heading
Heading = Clipboard("Employee Data")
```

The following statement replaces the selected text in the MultiLineEdit mle_terms with the contents of the clipboard:

```
mle_terms.ReplaceText(Clipboard())
```

The following statement exchanges the contents of the StaticText st_welcome with the contents of the clipboard:

```
st_welcome.Text = Clipboard(st_welcome.Text)
```

See also

Clear
Copy
Cut
Paste
Replace
ReplaceText

Syntax 2 For bitmaps of graphs

Description

Replaces the contents of the system clipboard with a bitmap image of a graph. You can paste the image into other applications.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Graph objects in windows and user objects, and graphs in DataWindow controls and DataStore objects

Syntax

name.**Clipboard** ({ *graphobject* })

Argument	Description
name	The name of the graph or the DataWindow control or DataStore containing the graph you want to copy to the clipboard
graphobject (DataWindow control and DataStore only)	A string whose value is the name of the graph in the DataWindow object that you want to copy to the clipboard

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is null, Clipboard returns null.

Examples

This statement copies the graph gr_products_data to the clipboard:

```
gr_products_data.Clipboard()
```

This statement copies the graph gr_employees in the DataWindow control dw_emp_data to the clipboard:

```
dw_emp_data.Clipboard("gr_employees")
```

Close

Closes a window, scanner, SMS or peripheral device connection, or a file that you opened with the FileDirect object.

To close	Use
A window	Syntax 1
A BarcodeScanner or BiometricScanner object	Syntax 2
A communications channel for a Camera object	Syntax 3
A communications channel for a SerialGPS object	Syntax 4
A Short Message Service (SMS) session	Syntax 5
A FileDirect object	Syntax 6
A trace file	Syntax 7

Syntax 1 For windows

Description

Closes a window and releases the storage occupied by the window and all the controls in the window.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Window objects

Syntax

Close (*windowname*)

Argument	Description
windowname	The name of the window you want to close

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs. If *windowname* is null, Close returns null. The return value is usually not used.

Usage

Use Syntax 1 to close a window and release the storage occupied by the window and all the controls in the window.

When you call Close, PocketBuilder removes the window from view, closes it, executes the scripts for the CloseQuery and Close events (if any), and then executes the rest of the statements in the script that called the Close function.

After a window is closed, its properties, instance variables, and controls can no longer be referenced in scripts. If a statement in the script references the closed window or its properties or instance variables, an execution error will result.

Preventing a window from closing
You can prevent a window from being closed with a return code of 1 in the script for the CloseQuery event. Use the RETURN statement.

Examples

These statements close the window w_employee and then open the window w_departments:

```
Close(w_employee)
Open(w_departments)
```

After you call Close, the following statements in the script for the CloseQuery event prompt the user for confirmation and prevent the window from closing:

```
IF MessageBox('ExitApplication', &
'Exit?', Question!, YesNo!) = 2 THEN
    // If no, stop window from closing
```

```
        RETURN 1
    END IF
```

See also

Hide
Open

Syntax 2 For BarcodeScanner and BiometricScanner objects

Description

Clears all buffers, detaches from scanner firmware, and unloads all DLLs.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✗

Applies to

BarcodeScanner and BiometricScanner objects

Syntax

Integer *scanner*.Close ( )

Argument	Description
scanner	The scanner object that you want to close

Return value

Integer. Returns 1 for success or -1 if an error occurs.

Usage

This is an optional method. It is always called by the Destructor event of BarcodeScanner and BiometricScanner objects.

Examples

The following closes scanner DLLs and disconnects from the scanner device firmware:

```
li_rtn = l_scanner.Close()
```

See also

Flush
Open

Syntax 3 For Camera objects

Description

Closes a communications channel for a camera if one is open and deactivates any data handlers.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✗

Applies to

Camera objects

Syntax

objectname.Close ( )

Argument	Description
objectname	Name of the Camera object

Return value

Integer. Returns 1 for success and a negative number for an error.

Usage

Use the Close function to close a communications channel for a Camera object you previously opened using the Open function.

Examples

The following script closes a file:

```
    li_ret = myCamera.close ( )
```

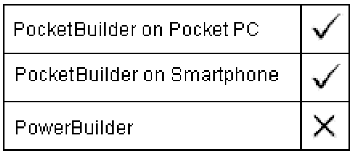
See also

Open

Syntax 4 For GPS and SerialGPS objects

Description

Closes a GPS communications channel if one is open and deactivates any data handlers.



PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✗

Applies to

SerialGPS objects

Syntax

GPSname.Close ( )

Argument	Description
GPSname	Name of the SerialGPS object

Return value

Integer. Returns 1 for success and a negative number for an error. The following is a list of possible error codes and their meanings:

-1 General error.

-10 Invalid object. Could occur if the GPS object instance is corrupted.

-13 Not previously opened. This function cannot be called until a GPS object or SerialGps object has been successfully opened.

Usage

Use the Close function to close a communications channel for a SerialGPS object you previously opened using the Open function. The Close function reinitializes all internal variables to their default values, but the ConfigParams property for SerialGPS objects is not reinitialized. This allows the user to reopen the SerialGPS object without having to respecify all of the configuration parameters.

Examples

The following script closes a file:

```
li_ret = myGPS.close ( )
```

See also

Open

Syntax 5 For SMSSession objects

Description

Closes a Short Message Service (SMS) session.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✗

Applies to

SMSSession objects

Syntax

SMSsessionname.Close ( )

Argument	Description
SMSsessionname	Name of the SMSSession object

Return value

Integer. Returns 1 for success and a negative value if an error occurs.

Usage

Use the Close function to close an SMS session you previously opened using the Open function.

Examples

The following script closes a file:

```
li_ret = mySMSSession.close ( )
```

See also

Open

Syntax 6 For FileDirect objects

Description

Closes a file that you open with the FileDirect object.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✗

Applies to

FileDirect objects

Syntax

instancename.Close ( )

Argument	Description
instancename	Instance name of the FileDirect object

Return value

Integer. Returns 1 for success and a negative number for an error.

Usage

Use the Close function to close a file you previously opened using the Open function.

Examples

The following script closes a file:

```
li_ret = nvo_FileDirect.close ( )
```

See also

Open

Syntax 7 For trace files

Description

Closes an open trace file.

PocketBuilder on Desktop	✓
PocketBuilder on Pocket PC	✗
PocketBuilder on Smartphone	✗
PowerBuilder	✓

Applies to

TraceFile objects

Syntax

instancename.**Close** ( )

Argument	Description
instancename	Instance name of the TraceFile object

Return value

ErrorReturn. Returns one of the following values:

- Success!—The function succeeded

- FileNotOpenError!—A trace file has not been opened

Usage

You use the Close function to close a trace file you previously opened with the Open function. You use the Close and Open functions as well as the properties and functions of the TraceFile object to access the contents of a trace file directly. You use these functions if you want to perform your own analysis of the tracing data instead of building a model with the Profiling or TraceTree object and the BuildModel function.

Examples

This example closes a trace file:

```
ift_file.Close()
DESTROY ift_file
```

See also

NextActivity
Open
Reset

CloseChannel

Description

Closes a DDE channel.

PocketBuilder	✗
PowerBuilder	✓

Syntax

CloseChannel (*handle* {, *windowhandle* })

Return value

Integer. Returns 1 if it succeeds. If an error occurs, CloseChannel returns a negative integer.

CloseTab

Description

Removes a tab page from a Tab control that was opened previously with the OpenTab or OpenTabWithParm function. CloseTab executes the scripts for the user object's Destructor event.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✓

Applies to

Tab controls

Syntax

tabcontrolname.**CloseTab** (*userobjectvar*)

Argument	Description
tabcontrolname	The name of the Tab control containing the tab page you want to close
userobjectvar	The name of the user object you want to close

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is null, CloseTab returns null. The return value is usually not used.

Usage

CloseTab closes a user object that has been opened as a tab page and releases the storage occupied by the object and its controls.

When you call CloseTab, PocketBuilder removes the tab page from the control, closes it, executes the script for the Destructor event (if any), and then executes the rest of the statements in the script that called the CloseTab function.

CloseTab also removes the user object from the Tab control's Control array, which is a property that lists the tab pages within the Tab control. If the closed tab page was not the last element in the array, the index for all subsequent tab pages is reduced by one.

After a user object is closed, its properties, instance variables, and controls can no longer be referenced in scripts. If a statement in the script references the closed user object or its properties or instance variables, an execution error will result.

Examples

These statements close the tab page user object u_employee and then open the user object u_departments in the Tab control tab_personnel:

```
tab_personnel.CloseTab(u_employee)
tab_personnel.OpenTab(u_departments)
```

When the user chooses a menu item that closes a user object, the following excerpt from the menu item's script prompts the user for confirmation before closing the u_employee user object in the window to which the menu is attached:

```
IF MessageBox("Close ", "Close?", &
    Question!, YesNo!) = 1 THEN
    // User chose Yes, close user object.
    ParentWindow.CloseTab(u_employee)
    // If user chose No, take no action.
END IF
```

See also

OpenTab

CloseUserObject

Description

Closes a user object by removing it from view and executing the scripts for its Destructor event.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Window objects

Syntax

windowname.**CloseUserObject** (*userobjectname*)

Argument	Description
windowname	The name of the window that contains the user object
userobjectname	The name of the user object you want to close

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is null, CloseUserObject returns null. The return value is usually not used.

Usage

Use CloseUserObject to close a user object and release the storage occupied by the object and its controls.

When you call CloseUserObject, PocketBuilder removes the object from view, closes it, executes the script for the Destructor event (if any), and then executes the rest of the statements in the script that called the CloseUserObject function.

CloseUserObject also removes the user object from the window's Control array, which is a property that lists the window's controls. If the closed user object was not the last element in the array, the index for all subsequent user objects is reduced by one.

After a user object is closed, its properties, instance variables, and controls can no longer be referenced in scripts. If a statement in the script references the closed user object or its properties or instance variables, an execution error will result.

Examples

These statements close the user object u_employee and then open the user object u_departments in the window w_personnel:

```
w_personnel.CloseUserObject(u_employee)
w_personnel.OpenUserObject(u_departments)
```

When the user chooses a menu item that closes a user object, the following excerpt from the menu item's script prompts the user for confirmation before closing the u_employee user object in the window to which the menu is attached:

```
IF MessageBox("Close ", "Close?", &
    Question!, YesNo!) = 1 THEN
    // User chose Yes, close user object.
    ParentWindow.CloseUserObject(u_employee)
    // If user chose No, take no action.
END IF
```

See also

OpenUserObject

CloseWithReturn

Description

Closes a window and stores a return value in the Message object. You should use CloseWithReturn only for response windows.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Window objects

Syntax

CloseWithReturn (*windowname*, *returnvalue*)

Argument	Description
windowname	The name of the window you want to close.
returnvalue	The value you want to store in the Message object when the window is closed. *Returnvalue* must be one of these datatypes: • String • Numeric • PowerObject

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is null, CloseWithReturn returns null. The return value is usually not used.

Usage

The purpose of CloseWithReturn is to close a response window and return information from the response window to the window that opened it. Use CloseWithReturn to close a window, release the storage occupied by the window and all the controls in the window, and return a value.

Just as with Close, CloseWithReturn removes a window from view, closes it, and executes the script for the CloseQuery and Close events, if any. Before executing the event scripts, it also stores *returnvalue* in the Message object. Then PocketBuilder executes the rest of the script that called the CloseWithReturn function.

After a window is closed, its properties, instance variables, and controls can no longer be referenced in scripts. If a statement in the script references the closed window or its properties or instance variables, an execution error results.

PocketBuilder stores *returnvalue* in the Message object properties according to its datatype. In the script that called CloseWithReturn, you can access the returned value by specifying the property of the Message object that corresponds to the return value's datatype.

Table 10-1: Message object properties where return values are stored

Return value datatype	Message object property
Numeric	Message.DoubleParm
PowerObject (such as a structure)	Message.PowerObjectParm
String	Message.StringParm

Returning several values as a structure
To return several values, create a user-defined structure to hold the values and access the PowerObjectParm property of the Message object in the script that opened the response window. The structure is passed by value so you can access the information even if the original variable has been destroyed.

Referencing controls
User objects and controls are passed by reference, not by value. You cannot use CloseWithReturn to return a reference to a control on the closed window (because the control no longer exists after the window is closed). Instead, return the value of one or more properties of that control.

Preventing a window from closing
You can prevent a window from being closed with a return code of 1 in the script for the CloseQuery event. Use a RETURN statement.

Examples

This statement closes the response window w_employee_response, returning the string emp_name to the window that opened it:

```
CloseWithReturn(Parent, "emp_name")
```

Suppose that a menu item opens one window if the user is a novice and another window if the user is experienced. The menu item displays a response window called w_signon to prompt for the user's experience level. The user types an experience level in the SingleLineEdit control sle_signon_id. The OK button in the response window passes the text in sle_signon_id back to the menu item script. The menu item script checks the StringParm property of the Message object and opens the desired window.

The script for the Clicked event of the OK button in the w_signon response window is a single line:

```
CloseWithReturn(Parent, sle_signon_id.Text)
```

The script for the menu item is:

```
string ls_userlevel

// Open the response window
Open(w_signon)

// Check text returned in Message object
ls_userlevel = Message.StringParm

IF ls_userlevel = "Novice" THEN
    Open(win_novice)
ELSE
    Open(win_advanced)
END IF
```

See also

Close
OpenSheetWithParm
OpenUserObjectWithParm
OpenWithParm

CollapseItem

Description

Collapses the specified item.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

TreeView controls

Syntax

treeviewname.**CollapseItem** (*itemhandle*)

Argument	Description
treeviewname	The TreeView control in which you want to collapse an item
itemhandle	The handle of the item you want to collapse

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs.

Usage

If there is only one level-one entry, you can use the RootTreeItem handle as the argument to collapse the tree so that only the top-level node is displayed. However, CollapseItem collapses only the current item, so that if the children of the top-level item were expanded when the tree was collapsed, they will still be expanded when the top-level item is expanded.

If there is more than one level-one entry, using the RootTreeItem handle as the argument collapses only the first level-one entry.

Examples

This example collapses an item in a TreeView control:

```
long ll_tvi
ll_tvi = tv_list.FindItem(currenttreeitem!, 0)
tv_list.CollapseItem(ll_tvi)
```

This example collapses the top-level item in a TreeView control that has only one level-one entry:

```
long ll_tvi
ll_tvi = tv_list.FindItem(roottreeitem!, 0)
tv_list.CollapseItem(ll_tvi)
```

See also

ExpandItem
ExpandAll
FindItem

CommandParm

Description

Retrieves the argument string, if any, that followed the program name when the application was executed.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

CommandParm ( )

Return value

String. Returns the application's argument string if it succeeds and the empty string ("") if it fails or if there were no arguments.

Usage

Command arguments can follow the program name in the command line of a Windows program item or in the Program Manager's Run response window. For example, when the user chooses File>Run in the Program Manager and enters:

```
MyAppl C:\EMPLOYEE\EMPLIST.TXT
```

CommandParm retrieves the string *C:\EMPLOYEE\EMPLIST.TXT*.

If the application's command line includes several arguments, CommandParm returns them all as a single string. You can use string functions, such as Mid and Pos, to parse the string.

You do not need to call CommandParm in the application's Open event. Use the *commandline* argument instead.

Examples

These statements retrieve the command line arguments and save them in the variable *ls_command_line*:

```
string ls_command_line
ls_command_line = CommandParm()
```

If the command line holds several arguments, you can use string functions to separate the arguments. This example stores a variable number of arguments, obtained with CommandParm, in an array. The code assumes each argument is separated by one space. For each argument, the Pos function searches for a space; the Left function copies the argument to the array; and Replace removes the argument from the original string so the next argument moves to the first position:

```
string ls_cmd, ls_arg[]
integer i, li_argcnt

// Get the arguments and strip blanks
// from start and end of string
ls_cmd = Trim(CommandParm())

li_argcnt = 1

DO WHILE Len(ls_cmd) > 0
    // Find the first blank
    i = Pos( ls_cmd, " ")

    // If no blanks (only one argument),
    // set i to point to the hypothetical character
    // after the end of the string
    if i = 0 then i = Len(ls_cmd) + 1
    // Assign the arg to the argument array.
    // Number of chars copied is one less than the
    // position of the space found with Pos
    ls_arg[li_argcnt] = Left(ls_cmd, i - 1)

    // Increment the argument count for the next loop
    li_argcnt = li_argcnt + 1

    // Remove the argument from the string
    // so the next argument becomes first
    ls_cmd = Replace(ls_cmd, 1, i, "")
LOOP
```

CommitTransaction

Description	Declares that the EAServer transaction associated with the calling thread should be committed.

PocketBuilder	✗
PowerBuilder	✓

Applies to	CORBACurrent objects
Syntax	*CORBACurrent*.**CommitTransaction** (*breportheuristics*)
Return value	Integer. Returns 0 if it succeeds or a negative value if an error occurs.

ConnectToNewObject

Description	Creates a new object in the specified server application and associates it with a PowerBuilder OLEObject variable. ConnectToNewObject starts the server application if necessary.

PocketBuilder	✗
PowerBuilder	✓

Applies to	OLEObject objects, OLETxnObject objects
Syntax	*oleobject*.**ConnectToNewObject** (*classname*)
Return value	Integer. Returns 0 if it succeeds or a negative value if an error occurs.

ConnectToNewRemoteObject

Description	Creates a new OLE object in the specified remote server application (if security on the server allows it) and associates the new object with a PowerBuilder OLEObject variable. ConnectToNewRemoteObject starts the server application if necessary.

PocketBuilder	✗
PowerBuilder	✓

Applies to	OLEObject objects
Syntax	*oleobject*.**ConnectToNewRemoteObject** (*hostname*, *classname*)
Return value	Integer. Returns 0 if it succeeds or a negative value if an error occurs.

ConnectToObject

Description	Associates an OLE object with a PowerBuilder OLEObject variable and starts the server application. The OLEObject variable and ConnectToObject are used for OLE automation, in which the PowerBuilder application asks the server application to manipulate the OLE object programmatically.

PocketBuilder	✗
PowerBuilder	✓

Applies to	OLEObject objects
Syntax	*oleobject*.**ConnectToObject** (*filename* {, *classname* })
Return value	Integer. Returns 0 if it succeeds or a negative value if an error occurs.

ConnectToRemoteObject

Description	Associates an OLE object with a PowerBuilder OLEObject variable and starts the server application.

PocketBuilder	✗
PowerBuilder	✓

Applies to	OLEObject objects

Syntax	*oleobject*.**ConnectToRemoteObject** (*hostname*, *filename* {, *classname* })
Return value	Integer. Returns 0 if it succeeds or a negative value if an error occurs.

ConnectToServer

Description

Connects a client application to a server component. The client application must call ConnectToServer before it can use a remote object on the server. This function applies to distributed applications only.

PocketBuilder	✗
PowerBuilder	✓

Applies to

Connection objects

Syntax

connection.**ConnectToServer** ( )

Return value

Long. Returns 0 if it succeeds or a negative value if an error occurs.

Copy

Description

Puts selected text or an OLE object on the clipboard. Copy does not change the source text or object.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

DataWindow, MultiLineEdit, SingleLineEdit, RichTextEdit, DropDownListBox, DropDownPictureListBox, OLE controls, and OLE DWObjects (objects within a DataWindow object that is within a DataWindow control)

Syntax

objectref.**Copy** ( )

Argument	Description
objectref	One of the following: • The name of the control containing the text you want to copy to the clipboard. • The name of the OLE control or the fully qualified name of a OLE DWObject within a DataWindow control that contains the object you want to copy to the clipboard. The fully qualified name for a DWObject has this syntax: *dwcontrol*.**Object**.*dwobjectname* If *objectref* is a DataWindow, text is copied from the edit control over the current row and column. If *objectref* is a DropDownListBox, its AllowEdit property must be true.

Return value

Integer

For edit controls, Copy returns the number of characters that were copied to the clipboard. If no text is selected in *objectref*, no characters are copied and Copy returns 0. If an error occurs, Copy returns -1. If *objectref* is null, Copy returns null.

Usage

To select text for copying, the user can use the mouse or keyboard. You can also call the SelectText function in a script.

To insert the contents of the clipboard into a control, use the Paste function.

Copy does not delete the selected text or OLE object. To delete the data, use the Clear or Cut function.

Examples

Assuming the selected text in mle_emp_address is Temporary Address, these statements copy Temporary Address from mle_emp_address to the clipboard and store 17 in *copy_amt*:

```
integer copy_amt
copy_amt = mle_emp_address.Copy()
```

See also

Clear
Clipboard
Cut
Paste
ReplaceText
SelectText

CopyRTF

Description

Returns the selected text, pictures, and input fields in a RichTextEdit control or RichText DataWindow as a string with rich text formatting. Bitmaps and input fields are included in the string.

PocketBuilder	✗
PowerBuilder	✓

Applies to

DataWindow controls, DataStore objects, and RichTextEdit controls

Syntax

rtename.**CopyRTF** ({ *selected* {, *band* } })

Return value

String. Returns the selected text as a string.

CopyRTF returns an empty string ("") if:

- There is no selection and *selected* is true
- An error occurs

Cos

Description

Calculates the cosine of an angle.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

Cos (*n*)

Argument	Description
n	The angle (in radians) for which you want the cosine

Return value

Double. Returns the cosine of *n*. If *n* is null, Cos returns null.

Examples

This statement returns 1:

```
Cos(0)
```

This statement returns .540302:

```
Cos(1)
```

This statement returns -1:

```
Cos(Pi(1))
```

See also

ACos
Pi
Sin
Tan
Cos method for DataWindows in the *DataWindow Reference* or online Help

Cpu

Description

Reports the amount of CPU time that has elapsed since the application started.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

Cpu ( )

Return value

Long. Returns the number of milliseconds of CPU time elapsed since the start of your PocketBuilder application.

Examples

These statements determine the amount of CPU time that elapsed while a group of statements executed:

```
// Declare ll_start and ll_used as long integers.
long ll_start, ll_used

// Set the start equal to the current CPU usage.
ll_start = Cpu()
... // Executable statements being timed

// Set ll_used to the number of CPU seconds
// that were used (current CPU time - start).
ll_used = Cpu() - ll_start
```

CreateDirectory

Description

Creates a directory.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

File system

Syntax

CreateDirectory (directoryname)

Argument	Description
directoryname	String for the name of the directory you want to create

Return value

Integer. Returns 1 if the function succeeds and -1 if an error occurs.

Examples

This example creates a new subdirectory in the current path and then makes the new subdirectory the current directory:

```
string  ls_path="my targets"
integer li_filenum
CreateDirectory ( ls_path )
li_filenum = ChangeDirectory( ls_path )
```

See also

GetCurrentDirectory
RemoveDirectory

CreateInstance

Creates an instance of a remote object running on a middle-tier server.

To create a remote object instance	Use
From a PowerBuilder client	Syntax 1
From within an EAServer or COM+ component	Syntax 2

Syntax 1 For creating an object instance on a remote server

Description

Creates an instance of a component running on EAServer. This function can be used to instantiate a remote object from a PowerBuilder client. In addition, it can be used within a component running on EAServer to instantiate another component running on a different server.

PocketBuilder	✗
PowerBuilder	✓

Applies to

Connection objects

Syntax

connection.**CreateInstance** (*objectvariable* {, *classname* })

Return value

Long. Returns 0 if it succeeds or a negative value if an error occurs.

Syntax 2 For creating a component instance on the current server

Description

Creates an instance of a component running on the current EAServer or COM+ server. This function is called from within a component instance running on EAServer or COM+.

PocketBuilder	✗
PowerBuilder	✓

Applies to

TransactionServer objects

Syntax

transactionserver.**CreateInstance** (*objectvariable* {, *classname* })

Return value

Long. Returns 0 if it succeeds or a negative value if an error occurs.

CreatePage

Description
Creates a tab page if it has not already been created.

PocketBuilder	✗
PowerBuilder	✓

Applies to
User objects used as tab pages

Syntax
userobject.**CreatePage** ( )

Return value
Integer. Returns one of the following values:1 if the page is successfully created and -1 if the page was already created or if it is not a tab page.

1 — The tab page was successfully created
0 — The tab page has already been created
-1 — The user object is not a tab page

Cut

Description
Deletes selected text from the specified control and stores it on the clipboard, replacing the clipboard contents with the deleted text or object.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to
DataWindow, MultiLineEdit, SingleLineEdit, DropDownListBox, DropDownPictureListBox, and OLE controls

Syntax
controlname.**Cut** ( )

Argument	Description
controlname	The name of the control containing the text or object to be cut. If *controlname* is a DataWindow, text is cut from the edit control over the current row and column. If *controlname* is a DropDownListBox, the AllowEdit property must be true.

Return value

Long.

For editable controls, Cut returns the number of characters that were cut from *controlname* and stored in the clipboard. If no text is selected, no characters are cut and Cut returns 0. If an error occurs, Cut returns -1. If *controlname* is null, Cut returns null.

Usage

To select text for deleting, the user can use the mouse or keyboard. You can also call the SelectText function in a script.

To insert the contents of the clipboard into a control, use the Paste function.

To delete selected text or an OLE object but not store it in the clipboard, use the Clear function.

Examples

Assuming the selected text in mle_emp_address is Temporary, this statement deletes Temporary from mle_emp_address, stores it in the clipboard, and returns 9:

```
mle_emp_address.Cut()
```

See also

Copy
Clear
Clipboard
DeleteItem
Paste

DataCount

Description

Reports the number of data points in the specified series in a graph.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Graph controls in windows and user objects, and graphs in DataWindow controls and DataStore objects

Syntax

controlname.**DataCount** ({ *graphcontrol*, } *seriesname*)

Argument	Description
controlname	The name of the graph in which you want the number of data points in a specific series, or the name of the DataWindow control or DataStore containing the graph
graphcontrol (DataWindow control or DataStore only)	The name of the graph in the DataWindow control or DataStore for which you want the data point count for the series
seriesname	A string whose value is the name of the series for which you want the number of data points

Return value

Long. Returns the number of data points in the specified series if it succeeds and -1 if an error occurs. If any argument's value is null, DataCount returns null.

Examples

These statements store in *ll_count* the number of data points in the series named Costs in the graph gr_product_data:

```
long ll_count
ll_count = gr_product_data.DataCount("Costs")
```

These statements store in *ll_count* the number of data points in the series named Salary in the graph gr_dept in the DataWindow control dw_employees:

```
long ll_count
ll_count = &
    dw_employees.DataCount("gr_dept", "Salary")
```

See also

AddSeries
InsertSeries
SeriesCount

DataSource

Description

Allows a RichTextEdit control to share data with a DataWindow and display the data in its input fields.

PocketBuilder	✗
PowerBuilder	✓

Applies to

RichTextEdit controls

Syntax

rtename.**DataSource** (*dwsource*)

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs.

Date

Converts DateTime, string, or numeric data to data of type date or extracts a date value from a blob. You can use one of several syntaxes, depending on the datatype of the source data.

To	Use
Extract the date from DateTime data or extract a date stored in a blob	Syntax 1
Convert a string to a date	Syntax 2
Combine numeric data into a date	Syntax 3

Platform information for Windows
To make sure you get correct return values for the year, you must verify that yyyy is the Short Date Style for year in the Regional Settings of the user's Control Panel. Your program can check this with the RegistryGet function.

If the setting is not correct, you can ask the user to change it manually or have the application change it (by calling the RegistrySet function). The user may need to reboot after the setting is changed.

Syntax 1 For DateTime data and blobs

Description

Extracts a date from a DateTime value or from a blob whose first value is a date or DateTime value.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

Date (*datetime*)

Argument	Description
datetime	A DateTime value or a blob in which the first value is a date or DateTime value. The rest of the contents of the blob is ignored. *Datetime* can also be an Any variable containing a DateTime or blob.

Return value

Date. Returns the date in *datetime* as a date. If *datetime* contains an invalid date or an incompatible datatype, Date returns 1900-01-01. If *datetime* is null, Date returns null.

Examples

After a value for the DateTime variable *ldt_StartDateTime* has been retrieved from the database, this example sets *ld_StartDate* equal to the date in *ldt_StartDateTime*:

```
DateTime ldt_StartDateTime
date ld_StartDate
ld_StartDate = Date(ldt_StartDateTime)
```

Assuming the value of a blob variable *ib_blob* contains a DateTime value beginning at byte 32, the following statement converts it to a date value:

```
date ld_date
ld_date = Date(BlobMid(ib_blob, 32))
```

See also

DateTime

Syntax 2 For strings

Description

Converts a string whose value is a valid date to a date value.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

Date (*string*)

Argument	Description
string	A string containing a valid date (such as January 1, 1998, or 12-31-99) that you want returned as a date. *Datetime* can also be an Any variable containing a string.

Return value

Date. Returns the date in *string* as a date. If *string* contains an invalid date or an incompatible datatype, Date returns 1900-01-01. If *string* is null, Date returns null.

Usage

Valid dates in strings can include any combination of day (1 to 31), month (1 to 12 or the name or abbreviation of a month), and year (2 or 4 digits). PocketBuilder assumes a 4-digit number is a year. Leading zeros are optional for month and day. The month, whether a name, an abbreviation, or a number, must be in the month location specified in the system setting for a date's format. If you do not know the system setting, use the standard datatype date format yyyy-mm-dd.

Date literals do not need to be converted with the Date function.

Examples

Example 1 These statements all return the date datatype for text expressing the date July 4, 1994 (1994-07-04). The system setting for a date's format is set with the month's position in the middle:

```
Date("1994/07/04")
Date("1994 July 4")
Date("04 July 1994")
```

Example 2 The following groups of statements check to be sure the date in sle_start_date is a valid date and display a message if it is not. The first version checks the result of the Date function to see if the date was valid. The second uses the IsDate function to check the text before using Date to convert it:

Version 1:

```
// Windows Control Panel date format is YY/MM/DD
date ld_my_date

ld_my_date = Date(sle_start_date.Text)
IF ld_my_date = Date("1900-01-01") THEN
    MessageBox("Error", "This date is invalid: " &
    + sle_start_date.Text)
END IF
```

Version 2:

```
date ld_my_date
```

```
IF IsDate(sle_start_date.Text) THEN
    ld_my_date = Date(sle_start_date.Text)
ELSE
    MessageBox("Error", "This date is invalid: " &
    + sle_start_date.Text)
END IF
```

See also

DateTime
IsDate
RelativeDate
RelativeTime
Date method for DataWindows in the *DataWindow Reference*

Syntax 3 For combining numbers into a date

Description

Combines numbers representing the year, month, and day into a date value.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

Date (*year*, *month*, *day*)

Argument	Description
year	The 4-digit year (-9999 to 9999) of the date
month	The 1- or 2-digit integer for the month (1 to 12) of the year
day	The 1- or 2-digit integer for the day (1 to 31) of the month

Return value

Date. Returns the date specified by the integers for *year*, *month*, and *day* as a date datatype. If any value is invalid (out of the range of values for dates), Date returns 1900-01-01. If any argument's value is null, Date returns null.

Examples

These statements use integer values to set *ld_my_date* to 1994-10-15:

```
date ld_my_date
ld_my_date = Date(1994, 10, 15)
```

See also

DateTime
DaysAfter
RelativeDate
RelativeTime

DateTime

Manipulates DateTime values. There are two syntaxes.

To	Use
Combine a date and a time value into a DateTime value	Syntax 1
Obtain a DateTime value that is stored in a blob	Syntax 2

Syntax 1 For creating DateTime values

Description

Combines a date value and a time value into a DateTime value.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

DateTime (*date* {, *time* })

Argument	Description
date	A value of type date.
time (optional)	A value of type time. If you omit *time*, PocketBuilder sets *time* to 00:00:00.000000 (midnight). If you specify *time*, only the hour portion is required.

Return value

DateTime. Returns a DateTime value based on the values in *date* and optionally *time*. If any argument's value is null, DateTime returns null.

Usage

DateTime data is used only for reading and writing DateTime values to and from a database. To use the date and time values in scripts, use the Date and Time functions to assign values to date and time variables.

Examples

These statements convert the date and time stored in *ld_OrderDate* and *lt_OrderTime* to a DateTime value that can be used to update the database:

```
DateTime ldt_OrderDateTime
date ld_OrderDate
time lt_OrderTime

ld_OrderDate = Date(sle_orderdate.Text)
lt_OrderTime = Time(sle_ordertime.Text)
ldt_OrderDateTime = DateTime( &
    ld_OrderDate, lt_OrderTime)
```

See also

Date
Time
DateTime method for DataWindows in the *DataWindow Reference*

Syntax 2 For extracting DateTime values from blobs

Description

Extracts a DateTime value from a blob.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

DateTime (*blob*)

Argument	Description
blob	A blob in which the first value is a DateTime value. The rest of the contents of the blob is ignored. *Blob* can also be an Any variable containing a blob.

Return value

DateTime. Returns the DateTime value stored in *blob*. If *blob* is null, DateTime returns null.

Usage

DateTime data is used only for reading and writing DateTime values to and from a database. To use the date and time values in scripts, use the Date and Time functions to assign values to date and time variables.

Examples

After assigning blob data from the database to *lb_blob*, the following example obtains the DateTime value stored at position 20 in the blob (the length you specify for BlobMid must be at least as long as the DateTime value but can be longer):

```
DateTime dt
dt = DateTime(BlobMid(lb_blob, 20, 40))
```

See also

Date
Time

Day

Description

Obtains the day of the month in a date value.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

Day (*date*)

Argument	Description
date	A date value from which you want the day

Return value

Integer. Returns an integer (1 to 31) representing the day of the month in *date*. If *date* is null, Day returns null.

Examples

These statements extract the day (31) from the date literal 1994-01-31 and set *li_day_portion* to that value:

```
integer li_day_portion
li_day_portion = Day(1994-01-31)
```

These statements check to be sure the date in sle_date is valid, and if so set *li_day_portion* to the day in the sle_date:

```
integer li_day_portion

IF IsDate(sle_date.Text) THEN
    li_day_portion = Day(Date(sle_date.Text))
ELSE
    MessageBox("Error", &
    "This date is invalid: " &
    + sle_date.Text)
END IF
```

See also

Date
IsDate
Month
Year
Day method for DataWindows in the *DataWindow Reference* or the online Help

DayName

Description

Determines the day of the week in a date value and returns the weekday's name.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

DayName (*date*)

Argument	Description
date	A date value for which you want the name of the day

Return value

String. Returns a string whose value is the weekday (Sunday, Monday, and so on) of *date*. If *date* is null, DayName returns null.

Usage

DayName returns a name in the language of the runtime files available on the machine where the application is run. If you have installed localized runtime files in the development environment or on a user's machine, then on that machine the name returned by DayName is in the language of the localized files.

Examples

These statements evaluate the date literal 1993-07-04 and set *day_name* to Sunday:

```
string day_name
day_name = DayName(1993-07-04)
```

These statements check to be sure the date in sle_date is valid, and if so set *day_name* to the day in sle_date:

```
string day_name

IF IsDate(sle_date.Text) THEN
    day_name = DayName(Date(sle_date.Text))
ELSE
    MessageBox("Error", &
    "This date is invalid: " &
    + sle_date.Text)
END IF
```

See also

Day
DayNumber
IsDate
DayName in the *DataWindow Reference*

DayNumber

Description

Determines the day of the week of a date value and returns the number of the weekday.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

DayNumber (*date*)

Argument	Description
date	The date value from which you want the number of the day of the week

Return value

Integer. Returns an integer (1-7) representing the day of the week of *date*. Sunday is day 1, Monday is day 2, and so on. If *date* is null, DayNumber returns null.

Examples

These statements evaluate the date literal 1990-01-31 and set *day_nbr* to 4 (January 31, 1990, was a Wednesday):

```
integer day_nbr
day_nbr = DayNumber(1990-01-31)
```

These statements check to be sure the date in sle_date is valid, and if so set *day_nbr* to the number of the day in the sle_date:

```
integer day_nbr

IF IsDate(sle_date.Text) THEN
    day_nbr = DayNumber(Date(sle_date.Text))
ELSE
    MessageBox("Error", &
    "This date is invalid: " &
    + sle_date.Text)
END IF
```

See also

Day
DayName
IsDate
DayNumber in the *DataWindow Reference*

DaysAfter

Description

Determines the number of days one date occurs after another.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

DaysAfter (*date1*, *date2*)

Argument	Description
date1	A date value that is the start date of the interval being measured
date2	A date value that is the end date of the interval

Return value

Long. Returns a long whose value is the number of days *date2* occurs after *date1*. If *date2* occurs before *date1*, DaysAfter returns a negative number. If any argument's value is null, DaysAfter returns null.

Examples

This statement returns 4:

```
DaysAfter(2002-12-20, 2002-12-24)
```

This statement returns -4:

```
DaysAfter(2002-12-24, 2002-12-20)
```

This statement returns 0:

```
DaysAfter(2003-12-24, 2003-12-24)
```

This statement returns 5:

```
DaysAfter(2003-12-29, 2004-01-03)
```

If you declare *date1* and *date2* date variables and assign February 16, 2003, to *date1* and April 28, 2003, to *date2* as follows:

```
date date1, date2

date1 = 2003-02-16
date2 = 2003-04-28
```

then each of the following statements returns 71:

```
DaysAfter(date1, date2)
DaysAfter(2003-02-16, date2)
DaysAfter(date1, 2003-04-28)
DaysAfter(2003-02-16, 2003-04-28)
```

See also

RelativeDate
RelativeTime
SecondsAfter
DaysAfter in the *DataWindow Reference*

DBHandle

Description

Reports the handle for your DBMS.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Transaction objects

Syntax

transactionobject.**DBHandle** ( )

Argument	Description
transactionobject	The current transaction object

Return value

UnsignedLong. Returns the handle for your DBMS. *Transactionobject* must exist, and the database must be connected. If *transactionobject* is null, DBHandle returns null. If *transactionobject* does not exist, an execution error occurs. If there is not enough memory to connect to your DBMS, DBHandle returns a negative number.

Usage

DBHandle returns a valid handle only if you are connected to the database. It is not able to determine if the database connection does not exist or has been lost.

Examples

For examples, search for DBHandle in online Help.

DebugBreak

Description

Suspends execution and opens the Debug window. This function is for use at design time only.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

DebugBreak ( )

Return value

None

Usage

Insert a call to the DebugBreak function into a script at a point at which you want to suspend execution and examine the application. Then enable just-in-time debugging and run the application in the development environment.

Just-in-time debugging
You turn on just-in-time debugging on the General page of the System Options dialog box that you open from the PocketBuilder Tools>System Options menu.

When PocketBuilder encounters the DebugBreak function, the Debug window opens showing the current context.

Examples

This statement tests whether a variable is null and opens the Debug window if it is:

```
IF IsNull(auo_ext) THEN DebugBreak()
```

Dec

Description

Converts a string to a decimal number or obtains a decimal value stored in a blob.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

Dec (*stringorblob*)

Argument	Description
stringorblob	A string whose value you want returned as a decimal value or a blob in which the first value is the decimal you want. The rest of the contents of the blob is ignored. *Stringorblob* can also be an Any variable containing a string or blob.

Return value

Decimal. Returns the value of *stringorblob* as a decimal. If *stringorblob* is not a valid PowerScript number or if it contains an incompatible datatype, Dec returns 0. If *stringorblob* is null, Dec returns null.

Examples

This statement returns 24.3 as a decimal datatype:

```
Dec("24.3")
```

This statement returns the contents of the SingleLineEdit sle_salary as a decimal number:

```
Dec(sle_salary.Text)
```

For an example of assigning and extracting values from a blob, see Real.

See also

Double
Integer
Long
Real

DecoderName

Description

Retrieves the short decoder name for the passed in ID value.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✗

Applies to

BarcodeScanner objects

Syntax

String *scanner.*DecoderName (*decoderID*)

Argument	Description
scanner	The scanner object for which you want to retrieve a decoder name
decoderID	Integer value of the decoder

Return value	String. Returns the short decoder name.
Usage	Call DecoderName to determine the type of bar code scanned by a particular decoder.
Examples	The following example returns the name of the scanner device with a decoder ID of 48:

```
ls_name = l_scanner.DecoderName(48)
```

The decoder ID of 48 corresponds to UPCE0, which is the value assigned to the variable *ls_name* in the preceding script.

See also
DeviceInfo
DeviceNames

DeleteCategory

Description
Deletes a category and the data values for that category from the category axis of a graph.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Graph controls in windows and user objects. Does not apply to graphs within DataWindow objects (because their data comes directly from the DataWindow).

Syntax
controlname.**DeleteCategory** (*categoryvalue*)

Argument	Description
controlname	The graph in which you want to delete a category.
categoryvalue	A value that is the category you want to delete from *controlname*. The value you specify must be the same datatype as the datatype of the category axis.

Return value
Integer. Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is null, DeleteCategory returns null.

Examples
These statements delete the category whose name is entered in the SingleLineEdit sle_delete from the graph gr_product_data:

```
string CategName
```

```
CategName = sle_delete.Text
gr_product_data.DeleteCategory(CategName)
```

See also

DeleteData
DeleteSeries

DeleteColumn

Description

Deletes a column.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

ListView controls

Syntax

listviewname.**DeleteColumn** (*index*)

Argument	Description
listviewname	The name of the ListView control from which you want to delete a column
index	The index number of the column you want to delete

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs.

Examples

This example deletes the second column in a ListView control:

```
lv_list.DeleteColumn(2)
```

See also

DeleteColumns

DeleteColumns

Description

Deletes all columns.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

ListView controls

Syntax

listviewname.**DeleteColumns** ( )

Argument	Description
listviewname	The name of the ListView control from which you want to delete all columns

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs.

Examples

This example deletes all columns in a ListView control:

```
lv_list.DeleteColumns()
```

See also

DeleteColumn

DeleteData

Description

Deletes a data point from a series of a graph. The remaining data points in the series are shifted left to fill the data point's category.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Graph controls in windows and user objects. Does not apply to graphs within DataWindow objects (because their data comes directly from the DataWindow).

Syntax

controlname.**DeleteData** (*seriesnumber*, *datapointnumber*)

Argument	Description
controlname	The name of the graph in which you want to delete a data value
seriesnumber	The number of the series containing the data value you want to delete from *controlname*
datapointnumber	The number of the data point containing the data you want to delete

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is null, DeleteData returns null.

Examples

These statements delete the data at data point 7 in the series named Costs in the graph gr_product_data:

```
integer SeriesNbr
```

```
// Get the number of the series.
SeriesNbr = gr_product_data.FindSeries("Costs")
gr_product_data.DeleteData(SeriesNbr, 7)
```

See also

AddData
DeleteCategory
DeleteSeries
FindSeries

DeleteItem

Deletes an item from a ListBox, DropDownListBox, ListView, or Toolbar control.

To delete an item from	Use
A ListBox or DropDownListBox control	Syntax 1
A ListView control	Syntax 2
A TreeView control	Syntax 3
A Toolbar control	Syntax 4

Syntax 1 For ListBox and DropDownListBox controls

Description

Deletes an item from the list of values for a list box control.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

ListBox, DropDownListBox, PictureListBox, and DropDownPictureListBox controls

Syntax

listboxname.**DeleteItem** (*index*)

Argument	Description
listboxname	The name of the ListBox, DropDownListBox, or PictureListBox from which you want to delete an item
index	The position number of the item you want to delete

Return value

Integer. Returns the number of items remaining in the list of values after the item is deleted. If an error occurs, DeleteItem returns -1. If any argument's value is null, DeleteItem returns null.

Usage

If the control's Sorted property is set, the order of the list is probably different from the order you specified when you defined the control. If you know the item's text, use FindItem to determine the item's index.

Examples

Assuming lb_actions contains 10 items, this statement deletes item 5 from lb_actions and returns 9:

```
lb_actions.DeleteItem(5)
```

These statements delete the first selected item in lb_actions:

```
integer li_Index
li_Index = lb_actions.SelectedIndex()
lb_actions.DeleteItem(li_Index)
```

This statement deletes the item "Personal" from the ListBox lb_purpose:

```
lb_purpose.DeleteItem( &
    lb_purpose.FindItem("Personal", 1))
```

See also

AddItem
FindItem
InsertItem
SelectItem

Syntax 2 For ListView controls

Description

Deletes the specified item from a ListView control.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

ListView controls

Syntax

listviewname.**DeleteItem** (*index*)

Argument	Description
listviewname	The name of the ListView control from which you want to delete an item
index	The index number of the item you want to delete

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs.

Examples

This example uses SelectedIndex to find the index of the selected ListView item and then deletes the corresponding item:

```
integer index
index = lv_list.selectedindex()
lv_list.DeleteItem(index)
```

See also

AddItem
FindItem
InsertItem
SelectItem
DeleteItems

Syntax 3 For TreeView controls

Description

Deletes an item from a control and all its child items, if any.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

TreeView controls

Syntax

treeviewname.**DeleteItem** (*itemhandle*)

Argument	Description
treeviewname	The name of the TreeView control from which you want to delete an item
itemhandle	The handle of the item you want to delete

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs.

Usage

If all items are children of a single item at the root level, you can delete all items in the TreeView with the handle for RootTreeItem as the argument for DeleteItem. Otherwise, you need to loop through the items at the first level.

Examples

This example deletes an item from a TreeView control:

```
long ll_tvi
ll_tvi = tv_list.FindItem(CurrentTreeItem!, 0)
tv_list.DeleteItem(ll_tvi)
```

This example deletes all items from a TreeView control when there are several items at the first level:

```
long tvi_hdl = 0
```

```
DO UNTIL tv_1.FindItem(RootTreeItem!, 0) = -1
    tv_1.DeleteItem(tvi_hdl)
LOOP
```

See also

AddItem
FindItem
InsertItem
SelectItem
DeleteItems

Syntax 4 For Toolbar controls

Description

Deletes a toolbar item from the toolbar control.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✗

Applies to

Toolbar controls

Syntax

Integer *controlname*.DeleteItem (*index*)

Argument	Description
controlname	The name of the toolbar control
index	Integer for the index of the item that you want to remove from the toolbar

Return value

Integer. Returns 1 for success and -1 if an error occurs.

Examples

The following example removes the second item from the toolbar:

```
li_rtn = tlbr_mytoolbar.DeleteItem(2)
```

See also

AddItem
InsertItem

DeleteItems

Description

Deletes all items from a ListView control.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

ListView controls

Syntax

listviewname.**DeleteItems** ( )

Argument	Description
listviewname	The name of the ListView control from which you want to delete all items

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs.

Examples

This example deletes all the items in a ListView control:

```
lv_list.DeleteItems()
```

See also

DeleteItem

DeleteLargePicture

Description

Deletes a picture from the large image list.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

ListView controls

Syntax

listviewname.**DeleteLargePicture** (*index*)

Argument	Description
listviewname	The name of the ListView control to which you want to delete a large picture from the image list
index	The index entry for the large picture you want to delete

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs.

Examples

This example deletes a large picture from a ListView control:

```
lv_list.DeleteLargePicture(1)
```

See also

DeleteLargePictures

DeleteLargePictures

Description

Deletes all large pictures from a ListView control.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

ListView controls

Syntax

listviewname.**DeleteLargePictures** ( )

Argument	Description
listviewname	The name of the ListView control from which you want to delete all pictures from the large picture image list

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs.

Examples

This example deletes all large pictures from a ListView control:

```
lv_list.DeleteLargePictures()
```

See also

DeleteLargePicture

DeletePicture

Description

Deletes a picture from the image list.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

PictureListBox and TreeView controls

Syntax *controlname*.**DeletePicture** (*index*)

Argument	Description
controlname	The control from which you want to delete a picture
index	The index number of the picture you want to delete from the TreeView control's image list

Return value Integer. Returns 1 if it succeeds and -1 if an error occurs.

Usage When you delete a picture from the image list for a control, all subsequent pictures in the list are renumbered to fill the gap. Because the picture index for an item does not change, the pictures for items that use the affected index numbers will change.

Examples This example deletes the sixth image from the image list:

```
tv_list.DeletePicture(6)
```

See also DeletePictures
AddPicture

DeletePictures

Description Deletes all pictures from an image list.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to PictureListBox and TreeView controls

Syntax *controlname*.**DeletePictures** ( )

Argument	Description
controlname	The control in which you want to delete all pictures from the image list

Return value Integer. Returns 1 if it succeeds and -1 if an error occurs.

Examples This example deletes all images from a TreeView control image list:

```
tv_list.DeletePictures()
```

See also DeletePicture
AddPicture

DeleteSeries

Description

Deletes a series and its data values from a graph.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Graph controls in windows and user objects. Does not apply to graphs within DataWindow objects (because their data comes directly from the DataWindow).

Syntax

controlname.**DeleteSeries** (*seriesname*)

Argument	Description
controlname	The graph in which you want to delete a series
seriesname	A string whose value is the name of the series you want to delete from *controlname*

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is null, DeleteSeries returns null.

Usage

The series in a graph are numbered consecutively, in the order they were added to the graph. When a series is deleted, the remaining series are renumbered.

Examples

This script for the SelectionChanged event of a DropDownListBox assumes that the list box lists the series in the graph gr_data. When the user chooses an item, DeleteSeries deletes the series from the graph and DeleteItem deletes the name from the list box:

```
string ls_name
ls_name = This.Text
gr_data.DeleteSeries(ls_name)
This.DeleteItem(This.FindItem(ls_name, 0))
```

See also

AddSeries
DeleteCategory
DeleteData
FindSeries

DeleteSmallPicture

Description

Deletes a small picture from a ListView control.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

ListView controls

Syntax

listviewname.**DeleteSmallPicture** (*index*)

Argument	Description
listviewname	The name of the ListView control from which you want to delete a small picture from the image list
index	The index number of the small picture you want to delete

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs.

Examples

This example deletes a small picture from a ListView control:

```
lv_list.DeleteSmallPicture(1)
```

See also

DeleteSmallPictures

DeleteSmallPictures

Description

Deletes all small pictures from a ListView control.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

ListView controls

Syntax

listviewname.**DeleteSmallPictures** ( )

Argument	Description
listviewname	The name of the ListView control from which you want to delete all small pictures

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs.

Examples

This example deletes all small pictures from a ListView control:

```
lv_list.DeleteSmallPictures()
```

See also

DeleteSmallPicture

DeleteStatePicture

Description

Deletes a state picture from a control.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

ListView and TreeView controls

Syntax

controlname.**DeleteStatePicture** (*index*)

Argument	Description
controlname	The name of the ListView or TreeView control from which you want to delete a picture from the state image list
index	The index number of the state picture you want to delete

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs.

Examples

This example deletes a state picture from a ListView control:

```
lv_list.DeleteStatePicture(1)
```

See also

DeleteStatePictures

DeleteStatePictures

Description

Deletes all state pictures from a control.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

ListView and TreeView controls

Syntax

controlname.**DeleteStatePictures** ( )

Argument	Description
controlname	The name of the ListView or TreeView control from which you want to delete all state pictures

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs.

Examples

This example deletes all state pictures from a ListView control:

```
lv_list.DeleteStatePictures()
```

See also

DeleteStatePicture

DestroyModel

Description

Destroys the current performance analysis or trace tree model.

PocketBuilder on Desktop	✓
PocketBuilder on Pocket PC	✗
PocketBuilder on Smartphone	✗
PowerBuilder	✓

Applies to

Profiling and TraceTree objects

Syntax

***instancename*.DestroyModel** ( )

Argument	Description
instancename	Instance name of the Profiling or TraceTree object

Return value

ErrorReturn. Returns one of the following values:

- Success!—The function succeeded
- ModelNotExistsError!—The function failed because no model exists

Usage

When you are finished with the performance analysis or trace tree model you created using the BuildModel function, you must call DestroyModel to destroy the model as well as all the objects associated with that model. The memory allocated to a model will not be released until the object is destroyed.

Examples

This example destroys the performance analysis model previously created using the BuildModel function:

```
lpro_model.DestroyModel()
DESTROY lpro_model
```

See also BuildModel

DeviceInfo

Description Gets information specific to a scanner device.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✗

Applies to BarcodeScanner objects

Syntax Integer *scanner.*DeviceInfo (*values* [ ])

Argument	Description
scanner	The scanner object associated with the scanner device for which you want to get device-specific parameters
values [ ]	An array of integer values of the scanner device that is passed by reference

Return value Integer. Returns 1 for success or one of the following negative values if an error occurs:

- **-1** Unspecified error
- **-2** Supporting DLL not loaded error
- **-3** Initialization error other than DLL not loaded
- **-4** Error in the passed in arguments
- **-5** Something in the object instance is inconsistent
- **-6** Call to the driver failed
- **-7** Error opening the specific scan device
- **-8** Error in the internal buffer allocation
- **-10** Low level device error
- **-100** Feature not implemented

Usage The information retrieved is for a specific device. For the Symbol and Socket scanner devices, the information passed to the values array is shown in Table 10-2.

Table 10-2: Symbol scanner device information

Array number	Device-specific information (how value is encoded)
value [1]	Hardware version (hiword/loword)
value [2]	Decoder version (hiword/loword)
value [3]	Physical device driver version (hiword/loword)
value [4]	Model device driver version (hiword/loword)
value [5]	C-API version (hiword/loword)
value [6]	Supports narrow beam width (bool)
value [7]	Supports aiming (bool)
value [8]	Supports scan direction reporting (bool)
value [9]	Supports remote feedback (bool)
value [10]	Reader type (enumerated values: 0 for laser bar code reader, 1 for contact wand bar code reader, or 3 for imager bar code reader)

Examples

The following example retrieves the major and minor version numbers of the physical device driver for the Symbol scanner. It displays them in a list box:

```
unsignedlong l_info[ ]
integer li_rtn
long ver_major, ver_minor
string stmp

li_rtn = l_scanner.DeviceInfo(l_info)
// physical device driver version
ver_major = INTHIGH( l_info[3] )
ver_minor = INTLOW( l_info[3] )
stmp = string(ver_major) + "." + string(ver_minor)
lb_res.AddItem("Physical device driver: " + stmp )
```

See also

DecoderName
DeviceNames

DeviceNames

Description

Gets the names for a scanner device.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✗

Applies to

BarcodeScanner objects

Syntax

Integer *scanner.*DeviceNames (*names*)

Argument	Description
scanner	The scanner object associated with the scanner device for which you want to get a name
names	A string value for scanner device names that is passed by reference in the following format: `device_name<tab>user_friendly_name<tab>port_name`

Return value

Integer. Returns 1 for success or one of the following negative values if an error occurs:

- **-1** Unspecified error
- **-2** Supporting DLL not loaded error
- **-3** Initialization error other than DLL not loaded
- **-4** Error in the passed in arguments
- **-5** Something in the object instance is inconsistent
- **-6** Call to the driver failed
- **-7** Error opening the specific scan device
- **-8** Error in the internal buffer allocation
- **-10** Low level device error
- **-100** Feature not implemented

Usage

Call DeviceNames to determine the attached scanner device or devices. The default scanner device name is SCN1 for a Symbol scanner, and an empty string for a Socket scanner. You can parse the *names* argument value on the position of the tab characters to obtain separate strings for the short device name, the user-friendly (long) device name, and the port name used by the scanner device.

Examples

The following example gets a string with the tab separated names for the scanner device and its port:

```
string l_names
li_rtn = l_scanner.DeviceNames(l_names)
```

See also

DecoderName
DeviceInfo

DirectoryExists

Description

Determines if the named directory exists.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

DirectoryExists (*directoryname*)

Argument	**Description**
directoryname	String for the name of the directory you want to verify as existing

Return value

Returns true if the directory exists. Returns false if the directory does not exist or if you pass a file name in the *directoryname* argument.

Usage

You can use this method before attempting to move a file or delete a directory using other file system methods.

Examples

This example determines if a directory exists before attempting to move a file to it; otherwise it displays a message box indicating that the path does not exist:

```
string  ls_path="monthly targets"

If DirectoryExists ( ls_path ) Then
    FileMove ("2000\may.csv", ls_path+"\may.csv" )
    MessageBox ("File Mgr", "File moved to "&
     + ls_path + ".")
Else
    MessageBox ("File Mgr", "Directory " + ls_path+&
     " does not exist" )
End If
```

See also

FileMove
GetCurrentDirectory
RemoveDirectory

DirList

Description

Populates a ListBox with a list of files. You can specify a path, a mask, and a file type to restrict the set of files displayed. If the window has an associated StaticText control, DirList can display the current drive and directory as well.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

ListBox, DropDownListBox, PictureListBox, and DropDownPictureListBox controls

Syntax

listboxname.**DirList** (*filespec*, *filetype* {, *statictext* })

Argument	Description
listboxname	The name of the ListBox control you want to populate.
filespec	A string whose value is the file pattern. This is usually a mask (for example, *.*INI* or *.*TXT*). If you include a path, it becomes the current drive and directory.
filetype	An unsigned integer representing one or more types of files you want to list in the ListBox. Types are: • 0 — Read/write files • 1 — Read-only files • 2 — Hidden files • 4 — System files • 16 — Subdirectories • 32 — Archive (modified) files • 16384 — Drives • 32768 — Exclude read/write files from the list To list several types, add the numbers associated with the types. For example, to list read-write files, subdirectories, and drives, use 0+16+16384 or 16400 for *filetype.*
statictext (optional)	The name of the StaticText in which you want to display the current drive and directory.

Return value

Boolean. Returns true if the search path is valid so that the ListBox is populated or the list is empty. DirList returns false if the ListBox cannot be populated (for example, *filespec* is a file, not a directory, or specifies an invalid path). If any argument's value is null, DirList returns null.

Usage

You can call DirList when the window opens to populate the list initially. You should also call DirList in the script for the SelectionChanged event to repopulate the list box based on the new selection. (See the example in DirSelect.)

Alternatives
Although DirList's features allow you to emulate the standard File Open and File Save windows, you can get the full functionality of these standard windows by calling GetFileOpenName and GetFileSaveName instead of DirList.

Examples

This statement populates the ListBox lb_emp with a list of read/write files with the file extension *TXT* in the search path *C:\EMPLOYEE\*.TXT*:

```
lb_emp.DirList("C:\EMPLOYEE\*.TXT", 0)
```

This statement populates the ListBox lb_emp with a list of read-only files with the file extension *DOC* in the search path *C:\EMPLOYEE\*.DOC* and displays the path specification in the StaticText st_path:

```
lb_emp.DirList("C:\EMPLOYEE\*.DOC", 1, st_path)
```

These statements in the script for a window Open event initialize a ListBox to all files in the current directory that match **.TXT*:

```
String s_filespec
s_filespec = "*.TXT"
lb_filelist.DirList(s_filespec, 16400, st_filepath)
```

See also

DirSelect

DirSelect

Description

When a ListBox has been populated with the DirList function, DirSelect retrieves the current selection and stores it in a string variable.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

ListBox, DropDownListBox, PictureListBox, and DropDownPictureListBox controls

Syntax

listboxname.**DirSelect** (*selection*)

Argument	Description
listboxname	The name of the ListBox control from which you want to retrieve the current selection. The ListBox must have been populated using DirList, and the selection must be a drive letter, a file, or the name of a directory.
selection	A string variable in which the selected path name will be put.

Return value

Boolean. Returns true if the current selection is a drive letter or a directory name (which can contain files and other directories) and false if it is a file (indicating the user's final choice). If any argument's value is null, DirSelect returns null.

Usage

Use DirSelect in the SelectionChanged event to find out what the user chose. When the user's selection is a drive or directory, use the selection as a new directory specification for DirList.

Examples

The following script for the SelectionChanged event for the ListBox lb_FileList calls DirSelect to test whether the user's selection is a file. If not, the script joins the directory name with the file pattern, and calls DirList to populate the ListBox and display the current drive and directory in the StaticText st_FilePath. If the current selection is a file, other code processes the file name:

```
string ls_filename, ls_filespec = "*.TXT"

IF lb_FileList.DirSelect(ls_filename) THEN
    //If ls_filename is not a file,
    //append directory to ls_filespec.
    ls_filename = ls_filename + ls_filespec
    lb_filelist.DirList(ls_filename, &
      16400, st_FilePath)
ELSE
    ... //Process the file.
END IF
```

See also

DirList

Disable

Description

Disables an item on a menu. The menu item is dimmed (its color is changed to the user's disabled text color, usually gray), and the user cannot select it.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Menu objects

Syntax

menuname.**Disable** ( )

Argument	Description
menuname	The name of the menu selection you want to deactivate (disable)

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs. If *menuname* is null, Disable returns null.

Equivalent syntax Setting the menu's Enabled property is the same as calling Disable.

menuname.Enabled = false

This statement:

```
m_appl.m_edit.Enabled = FALSE
```

is equivalent to:

```
m_appl.m_edit.Disable()
```

Examples

This statement disables the m_edit menu item on the menu m_appl:

```
m_appl.m_edit.Disable()
```

See also

Enable

DisableCommit

Description

Declares that a component's transaction updates are inconsistent and cannot be committed in their present state.

PocketBuilder	✗
PowerBuilder	✓

Applies to	TransactionServer objects
Syntax	*transactionserver*.**DisableCommit** ( )
Return value	Integer. Returns 1 if it succeeds and -1 if an error occurs.

DisconnectObject

Description: Releases any object that is connected to the specified OLEObject variable.

PocketBuilder	✗
PowerBuilder	✓

Applies to	OLEObject objects
Syntax	*oleobject*.**DisconnectObject** ( )
Return value	Integer. Returns 0 if it succeeds and a negative value if an error occurs.

DisconnectServer

Description: Disconnects a client application from a server application.

PocketBuilder	✗
PowerBuilder	✓

Applies to	Connection objects
Syntax	*connection*.**DisconnectServer** ( )
Return value	Long. Returns 0 if it succeeds and a negative value if an error occurs.

Usage: After disconnecting from the server application, the client application needs to destroy the Connection object.

DisconnectServer causes all remote objects and proxy objects created for the client connection to be destroyed.

Examples: In this example, the client application disconnects from the server application using the Connection object *myconnect*:

```
myconnect.DisconnectServer()
destroy myconnect
```

See also: ConnectToServer

Display

Description

Displays the appointment, contact, or task using the default display in Pocket Outlook or the window specified as an optional argument to the POOM Login function.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✗

Applies to

POOMAppointment, POOMContact, POOMTask objects

Syntax

Integer *objectname*.Display ( )

Argument	Description
objectname	The name of the POOMAppointment, POOMContact, or POOMTask object

Return value

Integer. Returns 1 for success and one of the following negative values if an error occurs:

-1 Unspecified error

-2 Cannot connect to the repository or a required internal subobject failed to connect to the repository

-3 Cannot log in to the repository

-4 Incorrect input argument

-5 Action cannot be performed

-6 The object identifier (OID) is not in the repository

-7 Feature is not implemented yet

-8 No matching entries found for the criteria

Examples

The following example displays the first appointment in the list of appointments:

```
POOMAppointment appt
DateTime  dt

// global variable g_poom
appt = g_poom.GetAppointment(1)
dt = appt.AppointmentStart
appt.display()
```

See also

GetAppointment
GetContact
GetTask

Double

Description

Converts a string to a double or obtains a double value that is stored in a blob.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

Double (*stringorblob*)

Argument	Description
stringorblob	A string whose value you want returned as a double or a blob in which the first value is the double value. The rest of the contents of the blob is ignored. *Stringorblob* can also be an Any variable containing a double or blob.

Return value

Double. Returns the contents of *stringorblob* as a double. If *stringorblob* is not a valid PowerScript number or if it contains a non-numeric datatype, Double returns 0. If *stringorblob* is null, Double returns null.

Usage

To distinguish between a string whose value is the number 0 and a string whose value is not a number, use the IsNumber function before calling the Double function.

Examples

This statement returns 24.372 as a double:

```
Double("24.372")
```

This statement returns the contents of the SingleLineEdit sle_distance as a double:

```
Double(sle_distance.Text)
```

After assigning blob data from the database to lb_blob, this example obtains the double value stored at position 20 in the blob (the length you specify for BlobMid must be at least as long as the value but can be longer):

```
double lb_num
lb_num = Double(BlobMid(lb_blob, 20, 40))
```

For an example of assigning and extracting values from a blob, see Real.

See also

Dec
Integer
Long
Real

DoVerb

Description

Requests the OLE server application to execute the specified verb for the OLE object in an OLE control or OLE DWObject.

PocketBuilder	✗
PowerBuilder	✓

Applies to

OLE controls and OLE DWObjects (objects within a DataWindow object that is within a DataWindow control)

Syntax

objectref.**DoVerb** (*verb*)

Return value

Integer. Returns 0 if it succeeds and a negative value if an error occurs.

If any argument's value is null, DoVerb returns null.

Drag

Description

Starts or ends the dragging of a control.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✓

Applies to

All controls except drawing objects (Lines, Ovals, Rectangles, and Rounded Rectangles)

Syntax

control.**Drag** (*dragmode*)

Argument	Description
control	The name of the control you want to drag or stop dragging

Argument	Description
dragmode	A value of the DragMode datatype indicating the action you want to take on control: • Begin! — Put *control* in drag mode • Cancel! — Stop dragging *control* but do not cause a DragDrop event • End! — Stop dragging *control* and if *control* is over a target object, cause a DragDrop event

Return value

Integer. For all controls except OLE controls, returns 1 if it succeeds and -1 if you try to nest drag events or try to cancel the drag when *control* is not in drag mode. The return value is usually not used. If any argument's value is null, Drag returns null.

Usage

To see the list of draggable controls, open the Browser. All the objects in the hierarchy below dragobject are draggable.

If you set the control's DragAuto property to true, PocketBuilder automatically puts the control in drag mode when the user clicks it. The user must hold the stylus (PocketBuilder) or mouse button (PowerBuilder) down to drag.

Windows CE platforms
Dragging controls is not a typical action for Pocket PC applications.

When you use Drag(Begin!) in a control's Clicked event to manually put the control in drag mode, the user can drag the control by moving the mouse without holding down the mouse button. Clicking the left mouse button ends the drag. CANCEL! and END! are required *only* if you want to end the drag without requiring the user to click the left mouse button.

Dragging DataWindow controls
The Clicked event of a DataWindow control occurs when the user presses the mouse button, not when the mouse button is released. If you place Drag(Begin!) in a DataWindow control's Clicked event, releasing the mouse button ends the drag.

To achieve the same behavior as with other controls, define a user-defined event for the DataWindow control called lbuttonup and map it to the pbm_lbuttonup event ID. Then place the following code in the lbuttonup event script (*ib_dragflag* is a boolean instance variable):

```
IF NOT ib_dragflag THEN
    this.Drag(Begin!)
    ib_dragflag = TRUE
ELSE
    ib_dragflag = FALSE
END IF
```

To make something happen when the user drags a control onto a target object, write scripts for one or more of the target's drag events (DragDrop, DragEnter, DragLeave, and DragWithin).

Examples

This statement puts sle_emp into drag mode:

```
sle_emp.Drag(Begin!)
```

See also

DraggedObject

DraggedObject

Description

Returns a reference to the control that triggered a drag event.

PocketBuilder	✗
PowerBuilder	✓

Obsolete function
Use the Drag event's source argument instead of calling the DraggedObject function.

Syntax

DraggedObject ( )

Return value

DragObject, a special datatype that includes all draggable controls (all the controls but no drawing objects). Returns a reference to the control that is currently being dragged.

Draw

Description

Draws a picture control at a specified location in the current window.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Picture controls

Syntax

picture.**Draw** (*xlocation*, *ylocation*)

Argument	Description
picture	The name of the picture control you want to draw in the current window
xlocation	The x coordinate of the location (in PowerBuilder units) at which you want to draw the picture
ylocation	The y coordinate of the location (in PowerBuilder units) at which you want to draw the picture

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is null, Draw returns null. The return value is usually not used.

Usage

Using the Draw function is faster and produces less flicker than successively changing the X property of a picture. This is because the Draw function draws directly on the window rather than recreating a small window with the picture in it for each change. Therefore, use Draw to draw pictures in animation.

To create animation, you can place a picture outside the visible portion of the window and then use the Draw function to draw it at different locations in the window. However, the image remains at all the positions where you draw it. If you change the position by small increments, each new drawing of the picture covers up most of the previous image.

Using Draw does not change the position of the picture control—it just displays the control's image at the specified location. Use the Move function to actually change the position of the control.

Examples

This statement draws the bitmap p_Train at the location specified by the X and Y coordinates 100 and 200:

```
p_Train.Draw(100, 200)
```

These statements draw the bitmap p_ Train in many different locations so it appears to move from left to right across the window:

```
integer horizontal
FOR horizontal = 1 TO 2000 STEP 8
    p_Train.Draw(horizontal, 100)
NEXT
```

See also
Move

DropCall

Description
Disconnects the current call.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✗

Applies to
PhoneCall objects

Syntax
objectname.DropCall ( )

Argument	Description
objectname	The name of the PhoneCall object whose current call will be disconnected

Return value
Integer. Returns 1 for success and a negative value if an error occurs.

Examples
The following statement disconnects the current call on pcall_1:

```
li_ret = pcall_1.AllowReceivingCalls(gb_Allow)
```

See also
AcceptCall
AllowReceivingCalls
MakeCall
SetHold
SetMute
SetRingTone

EditLabel

Put a label in a ListView or TreeView control into edit mode.

To enable editing of a label in a	Use
ListView control	Syntax 1
TreeView control	Syntax 2

Syntax 1 For editing a label in a ListView

Description

Puts a label in a ListView into edit mode.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

ListView controls

Syntax

listviewname.**EditLabel** (*index*)

Argument	Description
listviewname	The ListView control in which you want to enable label editing
index	The index of the ListView item to be edited

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs.

Usage

The EditLabels property for the ListView must be set to true to enable editing of labels. When this property is true, calling the EditLabel function sets focus on the item and enables editing. To disable editing when the user has finished editing the label, set the EditLabels property to false in the EndLabelEdit event.

If the EditLabels property is set to false, the EditLabel function does not enable editing.

Examples

This example allows the user to edit the label of the first selected item in the ListView control lv_1:

```
integer li_selected
li_selected = lv_1.SelectedIndex()
lv_1.EditLabels = TRUE
lv_1.EditLabel(li_selected)
```

See also

FindItem

Syntax 2 For editing a label in a TreeView

Description

Puts a label in a TreeView into edit mode.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

TreeView controls

Syntax

treeviewname.**EditLabel** (*itemhandle*)

Argument	Description
treeviewname	The TreeView control in which you want to enable label editing
itemhandle	The handle of the item to be edited

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs.

Usage

The EditLabels property for the TreeView must be set to true to enable editing of labels. When this property is true, calling the EditLabel function sets focus on the item and enables editing. To disable editing when the user has finished editing the label, set the EditLabels property to false in the EndLabelEdit event.

If the EditLabels property is set to false, the EditLabel function does not enable editing.

Examples

This example allows the user to edit the label of the current TreeView item:

```
long ll_tvi
ll_tvi = tv_list.FindItem(CurrentTreeItem!, 0)
tv_list.EditLabels = TRUE
tv_list.EditLabel(ll_tvi)
```

See also

FindItem

Enable

Description

Enables an item on a menu so a user can select it.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Menu objects

Syntax

menuname.**Enable** ( )

Argument	Description
menuname	The name of the menu selection you want to enable

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs. If *menuname* is null, Enable returns null.

Usage

Enabling a menu item changes its color to the active color (not the dimmed, or disabled, color). Calling Enable sets the item's Enabled property to true.

Equivalent syntax Setting the menu's Enabled property is the same as calling Enable.

menuname.Enabled = TRUE

This statement:

```
menu_appl.m_delete.Enabled = TRUE
```

is equivalent to:

```
menu_appl.m_delete.Enable()
```

Examples

This statement enables the m_delete menu selection on the menu m_appl:

```
m_appl.m_delete.Enable()
```

See also

Disable

EnableCommit

Description

Declares that a component's work may be incomplete but its transaction updates are consistent and can be committed.

PocketBuilder	✗
PowerBuilder	✓

Applies to

TransactionServer objects

Syntax

transactionserver.**EnableCommit** ( )

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs.

EnableDecoder

Description

Enables or disables a particular decodeer.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✗

Applies to

BarcodeScanner objects

Syntax

Integer *scanner.*EnableDecoder (*decoderID*, *fState*)

Argument	Description
scanner	The scanner object for which you want to enable or disable a decoder
decoderID	Integer value of the decoder you want to enable or disable
fState	Boolean value that determines whether to enable or disable the decoder

Return value

Integer. Returns 1 for success or one of the following negative values if an error occurs:

- **-1** Unspecified error
- **-2** Supporting DLL not loaded error
- **-3** Initialization error other than DLL not loaded
- **-4** Error in the passed in arguments
- **-5** Something in the object instance is inconsistent
- **-6** Call to the driver failed
- **-7** Error opening the specific scan device
- **-8** Error in the internal buffer allocation
- **-10** Low level device error
- **-11** Read is already pending (typically benign)
- **-100** Feature not implemented

Examples

The following example enables the decoder with a decoder ID of 49:

```
li_rtn = l_scanner.EnableDecoder(49, true)
```

The decoder ID of 49 corresponds to the decoder for UPCE1 type bar codes.

See also

DecoderName

EndPreview

Description

Ends preview mode for a camera device.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✗

Applies to

Camera objects

Syntax

objectname.EndPreview ( )

Argument	**Description**
objectname	The name of the camera object that you want to inquire about

Return value

Integer. Returns 1 for success, and one of the following negative values if an error occurs:

-1	Unspecified error
-2	Supporting DLL not loaded
-3	Other initialization error
-5	Inconsistency in this object instance
-6	Call to the driver or device failed
-7	Unsupported option
-8	Value for option is out of range

Examples

The following example ends the preview before closing the application:

```
li_rtn1 = g_myCamera.EndPreview()
li_rtn2 = g_camera.Close()
```

See also

BeginPreview
CaptureImage
GetAllowedImageAttributes
Open
SetCaptureImageAttributes
SetPreviewImageAttributes

EntryList

Description

Provides a list of the top-level entries included in a trace tree model.

PocketBuilder on Desktop	✓
PocketBuilder on Pocket PC	✗
PocketBuilder on Smartphone	✗
PowerBuilder	✓

Applies to

TraceTree objects

Syntax

instancename.**EntryList** (*list*)

Argument	Description
instancename	Instance name of the TraceTree object.
list	An unbounded array variable of datatype TraceTreeNode in which EntryList stores a TraceTreeNode object for each top-level entry in the trace tree model. This argument is passed by reference.

Return value

ErrorReturn. Returns the following values:

- Success!—The function succeeded
- ModelNotExistsError!—The function failed because no model exists

Usage

You use the EntryList function to extract a list of the top-level entries or nodes included in a trace tree model. Each top-level entry listed is defined as a TraceTreeNode object and provides the type of activity represented by that node.

You must have previously created the trace tree model from a trace file using the BuildModel function.

Examples

This example gets the top-level entries or nodes in a trace tree model and then loops through the list extracting information about each node. The of_dumpnode function takes a TraceTreeNode object and a level as arguments and returns a string containing information about the node:

```
TraceTree ltct_model
TraceTreeNode ltctn_list[], ltctn_node
Long ll_index,ll_limit
String ls_line

ltct_model = CREATE TraceTree
ltct_model.BuildModel()
ltct_model.EntryList(ltctn_list)
ll_limit = UpperBound(ltctn_list)
```

```
FOR ll_index = 1 TO ll_limit
   ltctn_node = ltctn_list[ll_index]
   ls_line += of_dumpnode(ltctn_node,0)
NEXT
...
```

See also BuildModel

ExecRemote

Asks a DDE server application to execute the specified command.

To send	Use
A single command to a DDE server application (a cold link)	Syntax 1
A command to a DDE server application after you have opened a channel (a warm link)	Syntax 2

Syntax 1 For sending single commands

Description Sends a single command to a DDE server application, called a **cold** link.

PocketBuilder	✗
PowerBuilder	✓

Syntax **ExecRemote** (*command*, *applname*, *topicname*)

Return value Integer. Returns 1 if it succeeds. If it fails, it returns a negative integer.

Syntax 2 For commands over an opened channel

Description Sends a command to a DDE server application when you have already called OpenChannel and established a warm link with the server.

PocketBuilder	✗
PowerBuilder	✓

Syntax **ExecRemote** (*command*, *handle* {, *windowhandle* })

Return value Integer. Returns 1 if it succeeds. If an error occurs, ExecRemote returns a negative integer.

Exp

Description

Raises *e* to the specified power.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

Exp (*n*)

Argument	Description
n	The power to which you want to raise *e* (2.71828)

Return value

Double. Returns *e* raised to the power *n*. If *n* is null, Exp returns null.

Inverse of Exp
The inverse of the Exp function is the Log function.

Examples

This statement returns 7.38905609893065.

```
Exp(2)
```

These statements convert a natural logarithm (base *e*) back to a regular number. When executed, Exp sets value to 200:

```
double value, x = log(200)
value = Exp(x)
```

See also

Log
LogTen
Exp method for DataWindows in the *DataWindow Reference*.

ExpandAll

Description

Recursively expands a specified item.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

TreeView controls

Syntax

treeviewname.**ExpandAll** (*itemhandle*)

Argument	Description
treeviewname	The TreeView control in which you want to expand an item and all the subordinate items in its hierarchy
itemhandle	The handle of the item you want to expand

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs.

Usage

To expand all levels in a TreeViewItem, use the ExpandAll function for the RootTreeItem.

Examples

This example expands all levels of a TreeView control:

```
long ll_tvi
ll_tvi = tv_list.FindItem(RootTreeItem! , 0)
tv_list.ExpandAll(ll_tvi)
```

See also

CollapseItem
ExpandItem
FindItem

ExpandItem

Description

Expands a specified item.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

TreeView controls

Syntax

treeviewname.**ExpandItem** (*itemhandle*)

Argument	Description
treeviewname	The TreeView control in which you want to expand an item
itemhandle	The handle of the item you want to expand

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs.

Usage

ExpandItem expands only a single item. To expand a specified item including its children, use ExpandAll.

Examples

This example expands the current level of a TreeView:

```
long ll_tvi
ll_tvi = tv_list.FindItem(CurrentTreeItem! , 0)
tv_list.ExpandItem(ll_tvi)
```

See also

CollapseItem
ExpandAll
FindItem

Fact

Description

Determines the factorial of a number.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

Fact (*n*)

Argument	Description
n	The number for which you want the factorial

Return value

Double. Returns the factorial of *n*. If *n* is null, Fact returns null.

Examples

This statement returns 24 (that is, 1 * 2 * 3 * 4):

```
Fact(4)
```

Both these statements return 1:

```
Fact(1)

Fact(0)
```

See also

Fact method for DataWindows in the *DataWindow Reference*

FARPrecedence

Description

Specifies whether the False Acceptance Rate (FAR) has precedence over the False Rejection Rate (FRR) for the results of a scanning operation.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✗

Applies to

BiometricScanner objects

Function availability
This function is not used by the HPBiometricScanner object implementation of the BiometricScanner base class.

Syntax

Integer *scanner*.FARPrecedence (*which*)

Argument	Description
scanner	The scanner object associated with the device you want to use to complete a scan
which	Boolean value for the precedence of the FAR. Values are:

Return value

Integer. Returns 1 for success or a negative values if an error occurs.

Usage

Setting the FAR precedence has meaning only if both the FAR and the FRR have nondefault values. This function is not used by the HPBiometricScanner object implementation.

Examples

The following example gives precedence to the FAR over the FRR:

```
li_rtn = l_bioscanner.FARPrecedence(true)
```

See also

MaxFARRequested
MaxFRRRequested

FileClose

Description

Closes the file associated with the specified file number. The file number was assigned to the file with the FileOpen function.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

FileClose (*file#*)

Argument	Description
file#	The integer assigned to the file you want to close. The FileOpen function returns the file number when it opens the file.

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs. If *file#* is null, FileClose returns null.

Examples

These statements open and then close the file *EMPLOYEE.DAT*. The variable *li_FileNum* stores the number assigned to the file when FileOpen opens the file. FileClose uses that number to close the file:

```
integer li_FileNum
li_FileNum = FileOpen("EMPLOYEE.DAT")
. . . // Some processing
FileClose(li_FileNum)
```

See also

FileLength
FileOpen
FileReadEx
FileWriteEx

FileCopy

Description

Copies one file to another, optionally overwriting the target file.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

FileCopy (*sourcefile*, *targetfile* {, *replace* })

Argument	Description
sourcefile	String for the name of the file you want to copy
targetfile	String for the name of the file you are copying to
replace (optional)	Boolean specifying whether to replace the target file (true) or not (false)

Return value

Integer. Returns values as follows:

1—Success
-1—Error opening *sourcefile*
-2—Error writing *targetfile*

Usage

If you do not specify a fully qualified path for *sourcefile* or for *targetfile*, the function works relative to the current directory. If you do not specify the *replace* argument, the FileCopy function does not replace a file in the target directory that has the same name as the name you specify in the *targetfile* argument (This is equivalent to setting the *replace* value to false).

Examples

The following example copies a file from the current directory to a different directory and saves the return value in a variable. It does not replace a file of the same name if one already exists in the target directory:

```
integer li_FileNum
li_FileNum = FileCopy ("jazz.gif" , &
   "C:\emusic\jazz.gif", FALSE)
```

See also

FileMove
GetCurrentDirectory

FileDelete

Description

Deletes the named file.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

FileDelete (*filename*)

Argument	Description
filename	A string whose value is the name of the file you want to delete

Return value

Boolean. Returns true if it succeeds, false if an error occurs. If *filename* is null, FileDelete returns null.

Examples

These statements delete the file the user selected in the Open File window:

```
integer ret, value
string docname, named

value = GetFileOpenName("Select File," &
       docname, named, "DOC", &
          "Doc Files (*.DOC),*.DOC")

IF value = 1 THEN ret = MessageBox("Delete", &
       "Delete file?", Question!, OKCancel!)
IF ret = 1 THEN FileDelete(docname)
```

See also

FileExists

FileExists

Description

Reports whether the specified file exists.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

FileExists (*filename*)

Argument	Description
filename	A string whose value is the name of a file

Return value

Boolean. Returns true if the file exists, false if it does not exist. If *filename* is null, FileExists returns null.

Usage

If *filename* is locked by another application, causing a sharing violation, FileExists also returns false.

Examples

This example determines if the file the user selected in the Save File window exists and, if so, asks the user if the file can be overwritten:

```
string ls_docname, ls_named
integer li_ret
boolean lb_exist
```

```
GetFileSaveName("Select File," ls_docname, &
       ls_named, "pkl", &
          "Doc Files (*.DOC),*.DOC")

lb_exist = FileExists(ls_docname)
IF lb_exist THEN li_ret = MessageBox("Save", &
       "OK to write over" + ls_docname, &
          Question!, YesNo!)
```

See also

FileDelete

FileLength

Description

Reports the length of a file whose size does not exceed 2GB in bytes.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

FileLength (*filename*)

Argument	Description
filename	A string whose value is the name of the file for which you want to know the length. If *filename* is not on the current application library search path, you must specify the fully qualified name.

Return value

Long. Returns the length in bytes of the file identified by *filename*. If the file does not exist, FileLength returns -1. If *filename* is null, FileLength returns null.

Usage

Call FileLength before or after you call FileOpen to check the length of a file before you call FileRead. The FileRead function can read a maximum of 32,765 characters at a time.

File security
If any security is set for the file (for example, if you are sharing the file on a network), you must call FileLength before FileOpen or after FileClose. Otherwise, you get a sharing violation.

The FileLength function cannot return the length of files whose size exceeds 2GB.

Examples

This statement returns the length of the file *EMPLOYEE.DAT* in the current directory:

```
FileLength("EMPLOYEE.DAT")
```

These statements determine the length of the *EMP.TXT* file in the *EAST* directory and open the file:

```
long LengthA
integer li_FileNum
LengthA = FileLength("C:\EAST\EMP.TXT")
li_FileNum = FileOpen("C:\EAST\EMP.TXT", &
      StreamMode!, Read!, LockReadWrite!)
```

See also

FileClose
FileOpen
FileRead
FileReadEx
FileWrite
FileWriteEx

FileMove

Description

Moves a file.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

FileMove (*sourcefile*, *targetfile*)

Argument	Description
sourcefile	String for the name of the file you want to move
targetfile	String for the name of the location you are moving the file

Return value

Integer. Returns values as follows:

1—Success
-1—Error opening *sourcefile*
-2—Error writing *targetfile*

Usage

You cannot write to a target file if a file with the same name already exists in the target directory. If you want to copy over a target file, you can use FileCopy and set the *replace* argument to true.

Examples

This example moves a file from the current directory to a different directory and saves the return value in the *li_FileNum* variable:

```
integer li_FileNum
li_FileNum = FileMove ("june.csv", &
   "H:/project/june2000.csv" )
```

See also

FileCopy
GetCurrentDirectory

FileOpen

Description

Opens the specified file for reading or writing and assigns it a unique integer file number. You use this integer to identify the file when you read, write, or close the file. The optional arguments *filemode*, *fileaccess*, *filelock*, and *writemode* determine the mode in which the file is opened.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

FileOpen (*filename* {, *filemode* {, *fileaccess* {, *filelock* {, *writemode* }}}})

Argument	Description
filename	A string whose value is the name of the file you want to open. If *filename* is not on the current directory's relative search path, you must enter the fully qualified name.
filemode (optional)	A value of the FileMode enumerated type that specifies how the end of a file read or file write is determined. Values are: • LineMode! — (Default) Read or write the file a line at a time • StreamMode! — Read the file in 32K chunks For more information, see Usage below.
fileaccess (optional)	A value of the FileAccess enumerated type that specifies whether the file is opened for reading or writing. Values are: • Read! — (Default) Read-only access • Write! — Write-only access If PocketBuilder does not find the file, a new file is created if the *fileaccess* argument is set to Write!

Argument	Description
filelock (optional)	A value of the FileLock enumerated type specifying whether others have access to the opened file. Values are: • LockReadWrite! — (Default) Only the user who opened the file has access • LockRead! — Only the user who opened the file can read it, but everyone has write access • LockWrite! — Only the user who opened the file can write to it, but everyone has read access • Shared! — All users have read and write access.
writemode (optional)	A value of the WriteMode enumerated datatype. When *fileaccess* is Write!, specifies whether existing data in the file is overwritten. Values are: • Append! — (Default) Write data to the end of the file • Replace! — Replace all existing data in the file *Writemode* is ignored if the *fileaccess* argument is Read!

Return value

Integer. Returns the file number assigned to *filename* if it succeeds and -1 if an error occurs. If any argument's value is null, FileOpen returns null.

Usage

The FileOpen function can open Unicode and ANSI files. If the file does not exist, FileOpen creates a Unicode file.

Creating valid Unicode files
A Unicode file that you create with FileOpen does not contain the Unicode byte order mark (BOM) at the beginning of the file when you set the *filemode* argument to StreamMode!. To include the BOM in a file that you open in stream mode, you can do one of the following:

- Create the file by calling FileOpen in line mode, add an innocuous character, such as a single space, then close the file and reopen it in stream mode using the Append! value for the *writemode* argument.
- Call FileWrite and pass the 2 byte binary blob "Char(65279)" before the rest of the string that you want to write to the file in stream mode.

The mode in which you open a file determines the behavior of the functions used to read and write to a file. There are two functions that read data from a file: FileRead and FileReadEx, and two functions that write data to a file: FileWrite and FileWriteEx. FileRead and FileWrite have limitations on the amount of data that can be read or written and are maintained for backward compatibility.

When a file has been opened in line mode, each call to the FileRead or FileReadEx function reads until it encounters a carriage return (CR), linefeed (LF), or end-of-file mark (EOF). Each call to FileWrite or FileWriteEx adds a CR and LF at the end of each string it writes. Line mode is not supported when reading from or writing to blobs.

When a file has been opened in stream mode, a call to FileRead reads the whole file (until it encounters an EOF) or 32,765 bytes, whichever is less. FileWrite writes a maximum of 32,765 bytes in a single call and does not add CR and LF characters. A byte size of 32,765 does not cause termination of the FileReadEx or FileWriteEx calls.

File not found
If PocketBuilder does not find the file, it creates a new file, giving it the specified name, if the *fileaccess* argument is set to Write!.

Examples

This example uses the default arguments and opens the file *EMPLOYEE.DAT* for reading. The default settings are LineMode!, Read!, and LockReadWrite!. FileReadEx reads the file line by line and no other user is able to access the file until it is closed:

```
integer li_FileNum
li_FileNum = FileOpen("EMPLOYEE.DAT")
```

This example opens the file *EMPLOYEE.DAT* in the *DEPT* directory in stream mode (StreamMode!) for write only access (Write!). Existing data is overwritten (Replace!). No other users can write to the file (LockWrite!):

```
integer li_FileNum
li_FileNum = FileOpen("C:\DEPT\EMPLOYEE.DAT", &
      StreamMode!, Write!, LockWrite!, Replace!)
```

See also

FileClose
FileLength
FileReadEx
FileWriteEx

FileRead

Description

Reads data from the file associated with the specified file number, which was assigned to the file with the FileOpen function. FileRead is maintained for backward compatibility. Use the FileReadEx function for new development.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

File format
FileRead can read Unicode files in line mode, and ANSI and Unicode files in stream mode. To read from an ANSI file in stream mode, use the FromANSI function to convert an ANSI blob into a Unicode character string.

Syntax

FileRead (*file#*, *variable*)

Argument	Description
file#	The integer assigned to the file when it was opened
variable	The name of the string or blob variable into which you want to read the data

Return value

Integer. Returns the number of characters or bytes read. If an end-of-file mark (EOF) is encountered before any characters are read, FileRead returns -100. If the file is opened in LineMode and a CR or LF is encountered before any characters are read, FileRead returns 0. If an error occurs, FileRead returns -1. If any argument's value is null, FileRead returns null. If the file length is greater than 32,765 bytes, FileRead returns 32,765.

Usage

If the file is opened in Line mode, FileRead reads a line of the file (that is, until it encounters a CR, LF, or EOF). It stores the contents of the line in the specified variable, skips the line-end characters, and positions the file pointer at the beginning of the next line.

If the file was opened in Stream mode, FileRead reads to the end of the file or the next 32,765 bytes, whichever is shorter. FileRead begins reading at the file pointer, which is positioned at the beginning of the file when the file is opened for reading. If the file is longer than 32,765 bytes, FileRead automatically positions the pointer after each read operation so that it is ready to read the next chunk of data.

FileRead can read a maximum of 32,765 characters at a time. Therefore, before calling the FileRead function, call the FileLength function to check the file length. If your system has file sharing or security restrictions, you may need to call FileLength before you call FileOpen.

An end-of-file mark is a null character (ASCII value 0). Therefore, if the file being read contains null characters, FileRead stops reading at the first null character, interpreting it as the end of the file.

Examples

This example reads the file *EMP_DATA.TXT* if it is short enough to be read with one call to FileRead:

```
integer li_FileNum
string ls_Emp_Input
long ll_FLength

ll_FLength = FileLength("C:\HR\EMP_DATA.TXT")
li_FileNum = FileOpen("C:\HR\EMP_DATA.TXT", &
      StreamMode!)
IF ll_FLength < 32767 THEN
      FileRead(li_FileNum, ls_Emp_Input)
END IF
```

This example reads the file *EMP_PIC1.BMP* and stores the data in the blob *Emp_Id_Pic*. The number of bytes read is stored in *li_bytes*:

```
integer li_fnum, li_bytes
blob Emp_Id_Pic

li_fnum = FileOpen("C:\HR\EMP_PIC1.BMP", &
      StreamMode!)
li_bytes = FileRead(li_fnum, Emp_Id_Pic)
```

This example reads a file exceeding 32,765 bytes. After the script has read the file into the blob *tot_b*, you can call the SetPicture or String function to make use of the data, depending on the contents of the file:

```
integer li_FileNum, loops, i
long flen, bytes_read, new_pos
blob b, tot_b

// Set a wait cursor
SetPointer(HourGlass!)

// Get the file length, and open the file
flen = FileLength(sle_filename.Text)
li_FileNum = FileOpen(sle_filename.Text, &
      StreamMode!, Read!, LockRead!)
```

```
// Determine how many times to call FileRead
IF flen > 32765 THEN
      IF Mod(flen, 32765) = 0 THEN
         loops = flen/32765
      ELSE
         loops = (flen/32765) + 1
      END IF
ELSE
      loops = 1
END IF

// Read the file
new_pos = 1

FOR i = 1 to loops
      bytes_read = FileRead(li_FileNum, b)
      tot_b = tot_b + b
NEXT

FileClose(li_FileNum)
```

See also

FileClose
FileLength
FileOpen
FileReadEx
FileSeek
FileWrite
FileWriteEx
FromAnsi

FileReadEx

Description

Reads data from the file associated with the specified file number, which was assigned to the file with the FileOpen function.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

FileReadEx (*file#*, *blob*)

FileReadEx (*file#*, *string*)

Argument	Description
file#	The integer assigned to the file when it was opened.
blob or *string*	The name of the string or blob variable into which you want to read the data.

Return value

Long. Returns the number of bytes read. If an end-of-file mark (EOF) is encountered before any characters are read, FileReadEx returns -100. If the file is opened in LineMode and a CR or LF is encountered before any characters are read, FileReadEx returns 0. If an error occurs, FileReadEx returns -1. FileReadEx returns -1 if you attempt to read from a string in stream mode or read from a blob in line mode. If any argument's value is null, FileReadEx returns null.

FileReadEx returns long
Unlike the FileRead function that it replaces, the FileReadEx function returns a long value.

Usage

If a file is opened in line mode, FileReadEx reads a line of the file until it encounters a CR, LF, or EOF. It stores the contents of the line in the specified variable, skips the line-end characters, and positions the file pointer at the beginning of the next line.

If the file was opened in stream mode, FileReadEx reads to the end of the file. FileReadEx begins reading at the file pointer, which is positioned at the beginning of the file when the file is opened for reading.

An end-of-file mark is a null character (ASCII value 0). Therefore, if the file being read contains null characters, FileReadEx stops reading at the first null character, interpreting it as the end of the file.

Examples

This example reads the file *EMP_DATA.TXT* into a string in line mode:

```
integer li_FileNum
string ls_Emp_Input

li_FileNum = FileOpen("C:\HR\EMP_DATA.TXT", &
   LineMode!)
   FileReadEx(li_FileNum, ls_Emp_Input)
END IF
```

This example reads the file *EMP_PIC1.BMP* and stores the data in the blob *Emp_Id_Pic*. The number of bytes read is stored in *ll_bytes*:

```
integer li_fnum
long ll_bytes
```

```
blob Emp_Id_Pic

li_fnum = FileOpen("C:\HR\EMP_PIC1.BMP", StreamMode!)
ll_bytes = FileReadEx(li_fnum, Emp_Id_Pic)
```

See also

FileClose
FileLength
FileOpen
FileRead
FileSeek
FileWriteEx

FileSeek

Description

Moves the file pointer to the specified position in a file whose size does not exceed 2GB. The file pointer is the position in the file at which the next read or write begins.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Effects of file format on character recognition
FileSeek can only be used in stream mode. It moves the pointer a specified number of bytes and does not take the size of Unicode characters into account. If you use FileSeek in a Unicode file and move the pointer to a position in the middle of a Unicode character, the character is not recognized and a FileRead from that position results in unexpected behavior.

Syntax

FileSeek (*file#*, *position*, *origin*)

Argument	Description
file#	The integer assigned to the file when it was opened.
position	A long whose value is the new position of the file pointer relative to the position specified in *origin*, in bytes.

Argument	Description
origin	The value of the SeekType enumerated datatype specifying where you want to start the seek. Values are: • FromBeginning! — (Default) At the beginning of the file • FromCurrent! — At the current position • FromEnd! — At the end of the file

Return value

Long. Returns the file position after the seek operation has been performed. If any argument's value is null, FileSeek returns null.

Usage

Use FileSeek to move within a binary file that you have opened in stream mode. FileSeek positions the file pointer so that the next FileRead or FileWrite occurs at that position within the file.

The FileSeek function cannot handle files whose size exceeds 2GB.

Examples

This example positions the file pointer 14 bytes from the end of the file:

```
integer li_FileNum
li_FileNum = FileOpen("emp_data")
FileSeek(li_FileNum, -14, FromEnd!)
```

This example moves the file pointer from its current position 14 bytes toward the end of the file. In this case, if no processing has occurred after FileOpen to affect the file pointer, specifying FromCurrent! is the same as specifying FromBeginning!:

```
integer li_FileNum
li_FileNum = FileOpen("emp_data")
FileSeek(li_FileNum, 14, FromCurrent!)
```

See also

FileRead
FileReadEx
FileWrite
FileWriteEx

FileWrite

Description

Writes data to the file associated with the specified file number. The file number was assigned to the file with the FileOpen function. FileWrite is maintained for backward compatibility. Use the FileWriteEx function for new development.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

File format
FileWrite can write to Unicode files in line mode, and ANSI and Unicode files in stream mode. To write to an ANSI file in stream mode, use the ToANSI function to convert a Unicode character string into an ANSI blob.

Syntax

FileWrite (*file#*, *variable*)

Argument	Description
file#	The integer assigned to the file when the file was opened
variable	A string or blob whose value is the data you want to write to the file

Return value

Integer. Returns the number of characters or bytes written if it succeeds and it returns -1 if an error occurs. If any argument's value is null, FileWrite returns null.

Usage

FileWrite writes its data at the position identified by the file pointer. If the file was opened with the *writemode* argument set to Replace!, the file pointer is initially at the beginning of the file. After each call to FileWrite, the pointer is immediately after the last write. If the file was opened with the *writemode* argument set to Append!, the file pointer is initially at the end of the file and moves to the end of the file after each write.

FileWrite sets the file pointer following the last character written. If the file was opened in line mode, FileWrite writes a carriage return (CR) and linefeed (LF) after the last character in *variable* and places the file pointer after the CR and LF.

Length limit
FileWrite can write only 32,766 bytes at a time, which includes the string terminator character. If the length of *variable* exceeds 32,765, FileWrite writes the first 32,765 characters and returns 32,765.

Examples

This script excerpt opens *EMP_DATA.TXT* and writes the string New Employees at the end of the file. The variable *li_FileNum* stores the number of the opened file:

```
integer li_FileNum
li_FileNum = FileOpen("C:\HR\EMP_DATA.TXT", &
      LineMode!, Write!, LockWrite!, Append!)
FileWrite(li_FileNum, "New Employees")
```

The following example reads a blob from the database and writes it to a file. The SQL SELECT statement assigns the picture data to the blob Emp_Id_Pic. Then FileOpen opens a file for writing in stream mode and FileWrite writes the blob to the file. You could use the Len function to test whether the blob was too big for a single FileWrite call:

```
integer li_FileNum
blob emp_id_pic

SELECTBLOB salary_hist
      INTO  : emp_id_pic
      FROM Employee
      WHERE Employee.Emp_Num = 100
      USING Emp_tran;

li_FileNum = FileOpen( &
      "C:\EMPLOYEE\EMP_PICS.BMP", &
         StreamMode!, Write!, Shared!, Replace!)
FileWrite(li_FileNum, emp_id_pic)
```

See also

FileClose
FileLength
FileOpen
FileRead
FileReadEx
FileSeek
FileWriteEx
ToAnsi

FileWriteEx

Description

Writes data to the file associated with the specified file number. The file number was assigned to the file with the FileOpen function.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

FileWriteEx (*file#*, *blob*)

FileWriteEx (*file#*, *string*)

Argument	Description
file#	The integer assigned to the file when the file was opened
blob or *string*	A blob or string whose value is the data you want to write to the file.

Return value

Long. Returns the number of bytes written if it succeeds and -1 if an error occurs. FileWriteEx returns -1 if you attempt to write to a string in stream mode or to a blob in line mode. If any argument's value is null, FileWriteEx returns null.

FileWriteEx returns long
Unlike the FileWrite function that it replaces, the FileWriteEx function returns a long value.

Usage

FileWriteEx writes its data at the position identified by the file pointer. If the file was opened with the *writemode* argument set to Replace!, the file pointer is initially at the beginning of the file. After each call to FileWriteEx, the pointer is immediately after the last write. If the file was opened with the *writemode* argument set to Append!, the file pointer is initially at the end of the file and moves to the end of the file after each write.

FileWriteEx sets the file pointer following the last character written. If the file was opened in line mode, FileWriteEx writes a carriage return (CR) and linefeed (LF) after the last character in *variable* and places the file pointer after the CR and LF.

If the file was opened in stream mode, FileWriteEx writes the full contents of the string or blob.

Examples

This script excerpt opens *EMP_DATA.TXT* and writes the string New Employees on a new line at the end of the file. The variable *li_FileNum* stores the number of the opened file:

```
integer li_FileNum
li_FileNum = FileOpen("C:\HR\EMP_DATA.TXT", &
      LineMode!, Write!, LockWrite!, Append!)
FileWriteEx(li_FileNum, "New Employees")
```

The following example reads a blob from the database and writes it to a file. The SQL SELECT statement assigns the picture data to the blob *Emp_Id_Pic*. Then FileOpen opens a file for writing in stream mode and FileWriteEx writes the blob to the file:

```
integer li_FileNum
blob emp_id_pic
SELECTBLOB salary_hist INTO  : emp_id_pic
   FROM Employee WHERE Employee.Emp_Num = 100
   USING Emp_tran;
li_FileNum = FileOpen("C:\EMPLOYEE\EMP_PICS.BMP", &
   StreamMode!, Write!, Shared!, Replace!)
FileWriteEx(li_FileNum, emp_id_pic)
```

See also

FileClose
FileLength
FileOpen
FileReadEx
FileSeek
FileWrite

Fill

Description

Builds a string of the specified length by repeating the specified characters until the result string is long enough.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

Fill (*chars*, *n*)

Argument	Description
chars	A string whose value will be repeated to fill the return string

Argument	Description
n	A long whose value is the length of the string you want returned

Return value

String. Returns a string *n* characters long filled with the characters in the argument *chars*. If the argument *chars* has more than *n* characters, the first *n* characters of *chars* are used to fill the return string. If the argument *chars* has fewer than *n* characters, the characters in *chars* are repeated until the return string has *n* characters. If any argument's value is null, Fill returns null.

Usage

Use Fill in printing routines to create a line or other special effect. For example, you can fill the amount line of a check with asterisks, or simulate a total line in a screen display by repeating hyphens below a column of figures.

Examples

This statement returns a string whose value is 35 stars:

```
Fill("*", 35)
```

This statement returns the string -+-+-+-:

```
Fill("-+", 7)
```

This statement returns 10 tildes (~):

```
Fill("~", 10)
```

See also

Space
Fill method for DataWindows in the *DataWindow Reference*

FillW

Description

Builds a string of the specified length by repeating the specified characters until the result string is long enough.

PocketBuilder	✗
PowerBuilder	✓

Obsolete function
This function is obsolete. It has the same behavior as Fill in SBCS and DBCS environments.

Syntax

FillW (*chars*, *n*)

Return value

String

Find

Description

Finds text in a RichTextEdit control or RichTextEdit DataWindow or DataStore.

PocketBuilder	✗
PowerBuilder	✓

For syntax for PocketBuilder DataWindows and DataStores, see the Find method for DataWindows in the *DataWindow Reference* or online Help.

Applies to

RichTextEdit controls and DataWindow controls (or DataStore objects) whose content has the RichTextEdit presentation style

Syntax

controlname.**Find** (*searchtext*, *forward*, *insensitive*, *wholeword*, *cursor*)

Return value

Integer. Returns the number of characters found. Find returns 0 if no matching text is found, and returns -1 if the DataWindow's presentation style is not RichTextEdit or an error occurs.

FindCategory

Description

Obtains the number of a category in a graph when you know the category's label.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Graph controls in windows and user objects, and graphs in DataWindow controls

Syntax

controlname.**FindCategory** ({ *graphcontrol*, } *categoryvalue*)

Argument	Description
controlname	A string whose value is the name of the graph in which you want to find a specific category, or the name of the DataWindow control containing the graph.
graphcontrol (DataWindow control only)	(Optional) A string whose value is the name of the graph in the DataWindow control in which you want to find a specific category.

Argument	Description
categoryvalue	A value that is the category for which you want the number. The value you specify must be the same datatype as the datatype of the category axis.

Return value

Integer. Returns the number of the category named in *categoryvalue* in the graph *controlname*, or if *controlname* is a DataWindow control, in *graphcontrol*. If an error occurs, FindCategory returns -1. If any argument's value is null, FindCategory returns null.

Usage

Most of the category manipulation functions require a category number, rather than a name. However, when you delete and insert categories, existing categories are renumbered to keep the numbering consecutive. Use FindCategory when you know only a category's label or when the numbering may have changed.

Examples

These statements obtain the number of a category in the graph gr_prod_data. The category name is the text in the SingleLineEdit sle_ctory:

```
integer CtgryNbr
CtgryNbr =gr_prod_data.FindCategory(sle_ctgry.Text)
```

These statements obtain the number of the category named Qty in the graph gr_computers in the DataWindow control dw_equip:

```
integer CtgryNbr
CtgryNbr = dw_equip.FindCategory("gr_computers", "Qty")
```

See also

AddCategory
DeleteData
DeleteSeries
FindSeries

FindClassDefinition

Description

Searches for an object in one or more PocketBuilder libraries (PKLs) or PowerBuilder libraries (PBLs) and provides information about its class definition.

PocketBuilder	✗
PowerBuilder	✓

PocketBuilder applications
This function can be used only in the development environment. It cannot be used in applications deployed to a Pocket PC or Smartphone device.

Syntax

FindClassDefinition (*classname* {, *librarylist* })

Argument	Description
classname	The name of an object (also called a class or class definition) for which you want information.
librarylist (optional)	An array of strings whose values are the fully qualified pathnames of PKLs or PBLs. If you omit *librarylist*, FindClassDefinition searches the library list associated with the running application.

Return value

ClassDefinition. Returns an object reference with information about the definition of *classname*. If any arguments are null, FindClassDefinition returns null.

Usage

There are two ways to get a ClassDefinition object containing class definition information:

- For an instantiated object in your application, use its ClassDefinition property
- For an object stored in a PKL or PBL, call FindClassDefinition

Examples

This example searches application libraries to find the class definition for w_mywindow:

```
ClassDefinition cd_windef
cd_windef = FindClassDefinition("w_mywindow")
```

This PowerBuilder example searches the libraries in the array *ls_libraries* to find the class definition for w_genapp_frame:

```
ClassDefinition cd_windef
string ls_libraries[ ]

ls_libraries[1] = "c:\pwrs\bizapp\windows.pbl"
ls_libraries[2] = "c:\pwrs\framewk\windows.pbl"
ls_libraries[3] = "c:\pwrs\framewk\ancestor.pbl"

cd_windef = FindClassDefinition(
      "w_genapp_frame", ls_libraries)
```

See also

FindFunctionDefinition
FindMatchingFunction

FindTypeDefinition

FindFunctionDefinition

Description

Searches for a global function in one or more PocketBuilder libraries (PKLs) or PowerBuilder libraries (PBLs) and provides information about the script definition.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

PocketBuilder applications
This function can be used only in the development environment. It cannot be used in applications deployed to a Pocket PC or Smartphone device.

Syntax

FindFunctionDefinition (*functionname* {, *librarylist* })

Argument	Description
functionname	The name of a global function for which you want information.
librarylist (optional)	An array of strings whose values are the fully qualified pathnames of PKLs or PBLs. If you omit *librarylist*, FindFunctionDefinition searches the library list associated with the running application.

Return value

ScriptDefinition. Returns an object reference with information about the script of *functionname*. If any arguments are null, FindFunctionDefinition returns null.

Usage

You can call FindClassDefinition to get a class definition for a global function. However, the ScriptDefinition object provides information tailored for functions.

Examples

This example searches the libraries for the running application to find the function definition for f_myfunction:

```
ScriptDefinition sd_myfunc
sd_myfunc = FindFunctionDefinition("f_myfunction")
```

This PowerBuilder example searches the libraries in the array *ls_libraries* to find the class definition for w_genapp_frame:

```
ScriptDefinition sd_myfunc
```

```
string ls_libraries[ ]

ls_libraries[1] = "c:\pwrs\bizapp\windows.pbl"
ls_libraries[2] = "c:\pwrs\framewk\windows.pbl"
ls_libraries[3] = "c:\pwrs\framewk\ancestor.pbl"

sd_myfunc = FindFunctionDefinition( &
      "f_myfunction", ls_libraries)
```

See also

FindClassDefinition
FindMatchingFunction
FindTypeDefinition

FindItem

Finds the next item in a list.

To find the next item	Use
In a ListBox, DropDownListBox, PictureListBox, or DropDownPictureListBox	Syntax 1
In a ListView control based upon its label	Syntax 2
By relative position in a ListView control	Syntax 3
By relative position in a TreeView control	Syntax 4

Syntax 1 For ListBox and DropDownListBox controls

Description

Finds the next item in a ListBox that begins with the specified search text.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

ListBox, DropDownListBox, PictureListBox, and DropDownPictureListBox controls

Syntax

listboxname.**FindItem** (*text*, *index*)

Argument	Description
listboxname	The name of the ListBox control in which you want to find an item.
text	A string whose value is the starting text of the item you want to find.

Argument	Description
index	The number of the item just before the first item to be searched. To search the whole list, specify 0.

Return value

Integer. Returns the index of the first matching item. To match, the item must start with the specified text; however, the text in the item can be longer than the specified text. If no match is found or if an error occurs, FindItem returns -1. If any argument's value is null, FindItem returns null.

Usage

When FindItem finds the matching item, it returns the index of the item but does not select (highlight) the item. To find *and* select the item, use the SelectItem function.

Examples

Assume the ListBox lb_actions contains the following list:

Index number	Item text
1	Open files
2	Close files
3	Copy files
4	Delete files

Then these statements start searching for Delete starting with item 2 (Close files). FindItem sets Index to 4:

```
integer Index
Index = lb_actions.FindItem("Delete", 1)
```

See also

AddItem
DeleteItem
InsertItem
SelectItem

Syntax 2 For ListView controls

Description

Searches for the next item whose label matches the specified search text.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

ListView controls

Syntax

listviewname.**FindItem** (*startindex, label, partial, wrap*)

Argument	Description
listviewname	The ListView control for which you want to search for items
startindex	The index number from which you want your search to begin
label	The string that is the target of the search
partial	If set to true, the search looks for a partial label match
wrap	If set to true, the search returns to the first index item after it has finished

Return value

Integer. Returns the index of the item found if it succeeds and -1 if an error occurs.

Usage

The search starts from *startindex* + 1 by default. To search from the beginning, specify 0.

If *partial* is set to true, the search string matches any label that begins with the specified text. If *partial* is set to false, the search string must match the entire label.

If *wrap* is set to true, the search wraps around to the first index item after searching to the end. If *wrap* is set to false, the search stops at the last index item in the ListView.

FindItem does not select the item it finds. You must use the item's selected property in conjunction with FindItem to select the resulting match.

Examples

This example takes the value from a SingleLineEdit control and passes it to FindItem:

```
listviewitem l_lvi
integer li_index
string ls_label

ls_label = sle_find.Text
IF ls_label = "" THEN
      MessageBox("Error" , &
         "Enter the name of a list item")
      sle_find.SetFocus()
ELSE
      li_index = lv_list.FindItem(0,ls_label,
TRUE,TRUE)
END IF
IF li_index = -1 THEN
      MessageBox("Error", "Item not found.")
ELSE
      lv_list.GetItem (li_index, l_lvi )
      l_lvi.HasFocus = TRUE
```

```
            l_lvi.Selected = TRUE
            lv_list.SetItem(li_index,l_lvi)
      END IF
```

See also

AddItem
DeleteItem
InsertItem
SelectItem

Syntax 3 For ListView controls

Description

Search for the next item relative to a specific location in the ListView control.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

ListView controls

Syntax

listviewname.**FindItem** (*startindex, direction, focused, selected, cuthighlighted, drophighlighted*)

Argument	Description
listviewname	The ListView control for which you want to search for items.
startindex	The index number from which you want your search to begin.
direction	The direction in which to search. Values are: DirectionAll! DirectionUp! DirectionDown! DirectionLeft! DirectionRight!
focused	If set to true, the search looks for the next ListView item that has focus.
selected	If set to true, the search looks for the next ListView item that is selected.
cuthighlighted	If set to true, the search looks for the next ListView item that is the target of a cut operation.
drophighlighted	If set to true, the search looks for next ListView item that is the target of a drag and drop operation.

Return value

Integer. Returns the index of the item found if it succeeds and -1 if an error occurs.

Usage

The search starts from *startindex* + 1 by default. If you want to search from the beginning, specify 0.

FindItem does not select the item it finds. You must use the item's selected property in conjunction with FindItem to select the resulting match.

If *focused*, *selected*, *cuthighlighted*, and *drophighlighted* are set to false, the search finds the next item in the ListView control.

Examples

This example uses FindItem to search from the selected ListView item:

```
listviewitem l_lvi
integer li_index li_startindex

li_startindex = lv_list.SelectedIndex()
li_index = lv_list.FindItem(li_startindex, &
      DirectionDown!, FALSE, FALSE ,FALSE, FALSE)

IF li_index = -1 THEN
      MessageBox("Error", "Item not found.")
ELSE
      lv_list.GetItem (li_index, l_lvi)
      l_lvi.HasFocus = TRUE
      l_lvi.Selected = TRUE
      lv_list.SetItem(li_index,l_lvi)
END IF
```

See also

AddItem
DeleteItem
InsertItem
SelectItem

Syntax 4 For TreeView controls

Description

Find an item based on its position in a TreeView control.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

TreeView controls

Syntax

treeviewname.**FindItem** (*navigationcode, itemhandle*)

Argument	Description
treeviewname	The name of the TreeView control in which you want to find a specified item.

Argument	Description
navigationcode	A value of the TreeNavigation enumerated datatype specifying the relationship between *itemhandle* and the item you want to find. See the table in Usage note for a list of valid values.
itemhandle	A long for the handle of an item related via *navigationcode* to the item for which you are searching.

Return value

Long. Returns the item handle if it succeeds and -1 if an error occurs.

Usage

FindItem does not select the item it finds. You must use the item's selected property in conjunction with FindItem to select the result of the FindItem search.

FindItem never finds a collapsed item, except when looking for ChildTreeItem!, which causes an item to expand. CurrentItem! is not changed until after the clicked event occurs. To return the correct handle for the current item when the user clicks it, create a custom event to return the handle and post it in the clicked event.

If *navigationcode* is RootTreeItem!, FirstVisibleTreeItem!, CurrentTreeItem!, or DropHighlightTreeItem!, set *itemhandle* to 0.

The following table shows valid values for the *navigationcode* argument.

Table 10-3: Valid values for the navigationcode argument of FindItem

Navigationcode value	What FindItem finds
RootTreeItem!	The first item at level 1. Returns -1 if no items have been inserted into the control.
NextTreeItem!	The sibling after *itemhandle*. A sibling is an item at the same level with the same parent. Returns -1 if there are no more siblings.
PreviousTreeItem!	The sibling before *itemhandle*. Returns -1 if there are no more siblings.
ParentTreeItem!	The parent of *itemhandle*. Returns -1 if the item is at level 1.
ChildTreeItem!	The first child of *itemhandle*. If the item is collapsed, ChildtreeItem! causes the node to expand. Returns -1 if the item has no children or if the item is not populated yet.
FirstVisibleTreeItem!	The first item visible in the control, regardless of level. The position of the scrollbar determines the first visible item.

Navigationcode value	What FindItem finds
NextVisibleTreeItem!	The next expanded item after *itemhandle*, regardless of level. The NextVisible and PreviousVisible values allow you to walk through all the visible children and branches of an expanded node. Returns -1 if the item is the last expanded item in the control. To scroll to an item that is beyond the reach of the visible area of the control, use FindItem and then SelectItem.
PreviousVisibleTreeItem!	The next expanded item before *itemhandle*, regardless of level. Returns -1 if the item is the first root item.
CurrentTreeItem!	The selected item. Returns -1 if the control never had focus and nothing has been selected.
DropHighlightTreeItem!	The item whose DropHighlighted property was most recently set. Returns -1 if the property was never set or if it has been set back to false because of other activity in the control.

Examples

To return the correct handle when the current item is clicked, place this code in a custom event that is posted in the item's clicked event:

```
long ll_tvi
ll_tvi = tv_list.FindItem(CurrentTreeItem!, 0)
```

This example finds the first item on the first level of a TreeView control:

```
long ll_tvi
ll_tvi = tv_list.FindItem(RootTreeItem!, 0)
```

See also

DeleteItem
GetItem
InsertItem
SelectItem

FindMatchingFunction

Description

Finds out what function in a class matches a specified signature. The signature is a combination of a script name and an argument list.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

PocketBuilder applications
This function can be used only in the development environment. It cannot be used in applications deployed to a Pocket PC or Smartphone device.

Applies to

ClassDefinition objects

Syntax

classdefobject.**FindMatchingFunction** (*scriptname*, *argumentlist*)

Argument	Description
classdefobject	The name of the ClassDefinition object describing the class in which you want to find a function.
scriptname	A string whose value is the name of the function.
argumentlist	An unbounded array of strings whose values are the datatypes of the function arguments. If the variable is passed by reference, the string must include "ref" before the datatype. If the variable is an array, you must include array brackets after the datatype. The format is: { ref } *datatype* { [] } For a bounded array, the argument must include the range, as in: `ref integer[1 TO 10]`

Return value

ScriptDefinition. Returns an object instance with information about the matching function. If no matching function is found, FindMatchingFunction returns null. If any argument is null, it also returns null.

Usage

In searching for the function, PocketBuilder examines the collapsed inheritance hierarchy. The found function may be defined in the current object or in any of its ancestors.

Arguments passed by reference   To find a function with an argument that is passed by reference, you must specify the REF keyword. If you have a VariableDefinition object for a function argument, check the CallingConvention argument to determine if the argument is passed by reference.

In documentation for PocketBuilder functions, arguments passed by reference are described as a variable, rather than simply a value. The PocketBuilder Browser does not report which arguments are passed by reference.

Examples

This example gets the ScriptDefinition object that matches the PowerBuilder window object function OpenUserObjectWithParm and looks for the version with four arguments. If it finds a match, the example calls the function uf_scriptinfo, which creates a report about the script:

```
string ls_args[]
ScriptDefinition sd

ls_args[1] = "ref dragobject"
ls_args[2] = "double"
ls_args[3] = "integer"
ls_args[4] = "integer"

sd = c_obj.FindMatchingFunction( &
       "OpenUserObjectWithParm", ls_args)
IF NOT IsValid(sd) THEN
       mle_1.Text = "No matching script"
ELSE
       mle_1.Text = uf_scriptinfo(sd)
END IF
```

The uf_scriptinfo function gets information about the function that matched the signature and builds a string. Scriptobj is the ScriptDefinition object passed to the function:

```
string s, lineend
integer li

lineend = "~r~n"

// Script name
s = s + scriptobj.Name + lineend
// datatype of the return value
s = s + scriptobj.ReturnType.DataTypeOf + lineend

// List argument names
s = s + "Arguments:" + lineend
FOR li = 1 to UpperBound(scriptobj.ArgumentList)
       s = s + scriptobj.ArgumentList[li].Name + lineend
NEXT

// List local variables
s = s + "Local variables:" + lineend
FOR li = 1 to UpperBound(scriptobj.LocalVariableList)
       s = s + scriptobj.LocalVariableList[li].Name &
           + lineend
NEXT
RETURN s
```

See also

FindClassDefinition
FindFunctionDefinition
FindTypeDefinition

FindNext

Description
Finds the next occurrence of text in the control and highlights it, using criteria set up in a previous call of the Find function.

PocketBuilder	✗
PowerBuilder	✓

Applies to
RichTextEdit controls and DataWindow controls whose content has the RichTextEdit presentation style

Syntax
controlname.**FindNext** ( )

Return value
Integer. Returns the number of characters found. FindNext returns 0 if no matching text is found and -1 if the DataWindow's presentation style is not RichTextEdit or an error occurs.

FindSeries

Description
Obtains the number of a series in a graph when you know the series' name.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to
Graph controls in windows and user objects, and graphs in DataWindow controls

Syntax
controlname.**FindSeries** ({ *graphcontrol*, } *seriesname*)

Argument	Description
controlname	The name of the graph containing the series for which you want the number, or the name of the DataWindow control containing the graph
graphcontrol (DataWindow control only)	(Optional) A string whose value is the name of the graph in the DataWindow control containing the series
seriesname	A string whose value is the name of the series for which you want the number

Return value

Integer. Returns the number of the series named in *seriesname* in the graph *controlname*, or if *controlname* is a DataWindow control, in *graphcontrol*. If an error occurs, FindSeries returns -1. If any argument's value is null, FindSeries returns null.

Usage

Most of the series manipulation functions require a series number, rather than a name. However, when you delete and insert series, existing series are renumbered so that the series are numbered consecutively. Use FindSeries when you know only a series' name or when the numbering may have changed.

Examples

These statements store the number of the series in the graph gr_product_data that was entered in the SingleLineEdit sle_series in *SeriesNbr*:

```
integer SeriesNbr
SeriesNbr = &
      gr_product_data.FindSeries(sle_series.Text)
```

These statements obtain the number of the series named PCs in the graph gr_computers in the DataWindow control dw_equipment and store it in *SeriesNbr*:

```
integer SeriesNbr
SeriesNbr = &
      dw_equipment.FindSeries("gr_computers", "PCs")
```

See also

AddSeries
DeleteSeries
FindCategory

FindTypeDefinition

Description

Searches for a type in one or more PocketBuilder libraries (PKLs) or PowerBuilder libraries (PBLs) and provides information about its type definition. You can also get type definitions for system types.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

PocketBuilder applications
This function can be used only in the development environment. It cannot be used in applications deployed to a Pocket PC or Smartphone device.

Syntax

FindTypeDefinition (*typename* {, *librarylist* })

Argument	**Description**
typename	The name of a simple datatype, enumerated datatype, or class for which you want information. To find a type definition for a nested type, use this form: *libraryEntryName`typename*
librarylist (optional)	An array of strings whose values are the fully qualified pathnames of PKLs or PBLs. If you omit *librarylist*, FindTypeDefinition searches the library list associated with the running application. PocketBuilder also searches its own libraries for built-in definitions, such as enumerated datatypes and system classes.

Return value

TypeDefinition. Returns an object reference with information about the definition of *typename*. If any arguments are null, FindTypeDefinition returns null.

Usage

The returned TypeDefinition object is a ClassDefinition, SimpleTypeDefinition, or EnumerationDefinition object. You can test the Category property to find out which one it is.

If you want to get information for a class, call FindClassDefinition instead. The arguments are the same and you are saved the step of checking that the returned object is a ClassDefinition object.

If you want to get information for a global function, call FindFunctionDefinition.

Examples

This example gets a TypeDefinition object for the grGraphType enumerated datatype. It checks the category of the type definition and, since it is an enumeration, assigns it to an EnumerationDefinition object type and saves the name in a string:

```
TypeDefinition td_graphtype
EnumerationDefinition ed_graphtype
string enumname

td_graphtype = FindTypeDefinition("grgraphtype")
IF td_graphtype.Category = EnumeratedType! THEN
      ed_graphtype = td_graphtype
      enumname = ed_graphtype.Enumeration[1].Name
END IF
```

This example is a function that takes a definition name as an argument. The argument is typename. It finds the named TypeDefinition object, checks its category, and assigns it to the appropriate definition object:

```
TypeDefinition td_def
SimpleTypeDefinition std_def
EnumerationDefinition ed_def
ClassDefinition cd_def

td_def = FindTypeDefinition(typename)
CHOOSE CASE td_def.Category
CASE SimpleType!
      std_def = td_def
CASE EnumeratedType!
      ed_def = td_def
CASE ClassOrStructureType!
      cd_def = td_def
END CHOOSE
```

This PowerBuilder example searches the libraries in the array *ls_libraries* to find the class definition for w_genapp_frame:

```
TypeDefinition td_windef
string ls_libraries[ ]

ls_libraries[1] = "c:\pwrs\bizapp\windows.pbl"
ls_libraries[2] = "c:\pwrs\framewk\windows.pbl"
ls_libraries[3] = "c:\pwrs\framewk\ancestor.pbl"

td_windef = FindTypeDefinition(
      "w_genapp_frame", ls_libraries)
```

See also

FindClassDefinition
FindFunctionDefinition
FindMatchingFunction

Flush

Description

Clears a scanner's internal buffers without detaching from scanner firmware or unloading scanner DLLs.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✗

Applies to

BarcodeScanner objects

Syntax

Integer *scanner*.Flush ( )

Argument	**Description**
scanner	The scanner object that is associated with the scanner you want to flush

Return value

Integer. Returns 1 for success or one of the following negative values if an error occurs:

- **-1** Unspecified error
- **-2** Supporting DLL not loaded error
- **-3** Initialization error other than DLL not loaded
- **-4** Error in the passed in arguments
- **-5** Something in the object instance is inconsistent
- **-6** Call to the driver failed
- **-7** Error opening the specific scan device
- **-8** Error in the internal buffer allocation
- **-9** Incorrect scan state for the requested action (typically benign)
- **-10** Low level device error
- **-100** Feature not implemented

Usage

Use the Flush function to make sure the scanner buffers are clear after aborted scans.

Examples

The following example flushes the internal buffers of the scanner device associated with the BarcodeScanner object:

```
li_rtn = l_scanner.Flush()
```

See also

Close

FocusToPreviousInstance

Description

Brings a running instance of a named window to the front of the current display.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✗

Syntax

FocusToPreviousInstance (*WindowTitle*)

Argument	Description
WindowTitle	A string identifying the title of the window that you want to bring to the front of the current display

Return value

Boolean. Returns true if it succeeds and false if it fails.

FromAnsi

Description

Converts a blob containing an ANSI character string to a Unicode string.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

FromAnsi (*blob*)

Argument	Description
blob	A blob containing an ANSI character string you want to convert to a Unicode string

Return value

String. Returns a character string if it succeeds and an empty string if it fails.

Usage

The FromAnsi function converts an ANSI character string contained in a blob to a Unicode character string.

Unicode file format
Unicode files sometimes have two extra bytes at the start of the file to indicate that they are Unicode files. If the two bytes are missing, PocketBuilder assumes “little endian” format.

Examples

This example reads a blob containing an ANSI character string from a file called *ansi.txt* and converts it into a string:

```
integer  li_filenum
blob  lb_text
string  ls_native

li_filenum = FileOpen("ansi.txt", StreamMode!)
FileRead(li_filenum, lb_text)
ls_native = FromAnsi(lb_text)
FileClose(li_filenum)
```

See also

FromUnicode
String
ToAnsi
ToUnicode

FromUnicode

Description

Converts a blob containing a Unicode character string to a string in the file format of the current version of PocketBuilder.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

FromUnicode (*blob*)

Argument	Description
blob	A blob containing a Unicode character string you want to convert to a string in the file format of the current version of PocketBuilder

Return value

String. Returns a character string if it succeeds and an empty string if it fails.

Usage

The FromUnicode function converts a Unicode blob to a Unicode character string and has the same result as String(*blob*). This function will be obsolete in a future release of PocketBuilder.

Unicode file format
Unicode files sometimes have two extra bytes at the start of the file to indicate that they are Unicode files. If the two bytes are missing, PocketBuilder assumes “little endian” format. If you are opening a Unicode file in stream mode, skip the first two bytes if they are present.

Examples

This example converts a Unicode blob that contains the definition of a window into a Unicode string.

```
integer li_fileone, li_filetwo
blob lb_text
string ls_native

li_fileone = FileOpen("D:\tst\w_one.srw", StreamMode!)

// Move the file pointer so that Unicode
// identifying characters aren't copied
FileSeek(li_Fileone, 2)

// Read the data in the file into a blob
FileRead(li_fileone, lb_text)
FileClose(li_fileone)

// Convert the Unicode blob to a string
ls_native = FromUnicode(lb_text)

// Open a second file to copy the string to
li_filetwo = FileOpen("w_one.srw", &
       StreamMode!, Write!)

FileWrite(li_filetwo, ls_native)
FileClose(li_filetwo)
```

See also

FromAnsi
ToAnsi
ToUnicode

GarbageCollect

Description	Forces immediate garbage collection.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax	**GarbageCollect** ( )
Return value	None
Usage	Forces garbage collection to occur immediately. PocketBuilder makes a pass to identify unused objects, including those with circular references, then deletes unused objects and classes.
Examples	This statement initiates garbage collection:

```
GarbageCollect()
```

See also	GarbageCollectGetTimeLimit GarbageCollectSetTimeLimit

GarbageCollectGetTimeLimit

Description	Gets the current minimum interval for garbage collection.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax	**GarbageCollectGetTimeLimit** ( )
Return value	Long. Returns the current minimum garbage collection interval.
Usage	Reads the current minimum period between garbage collection passes.
Examples	This statement returns the interval between garbage collection passes in the variable CollectTime:

```
long CollectTime

CollectTime = GarbageCollectGetTimeLimit()
```

See also	GarbageCollect GarbageCollectSetTimeLimit

GarbageCollectSetTimeLimit

Description

Sets the minimum interval between garbage collection passes.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

GarbageCollectSetTimeLimit (*newtimeinmilliseconds*)

Argument	Description
newtimeinmilliseconds	A long (in milliseconds) that you want to set as the minimum period between garbage collection cycles. If null, the existing interval is not changed.

Return value

Long. Returns the interval that existed before this function was called. If *newTime* is null, then null is returned and the current interval is not changed.

Usage

Specifies the minimum interval between garbage collection passes: garbage collection passes will not happen before this interval has expired.

Garbage collection can effectively be disabled by setting the minimum limit to a very large number. If garbage collection is disabled, unused classes will not be flushed out of the class cache.

Examples

This example sets the interval between garbage collection passes to 1 second and sets the variable *OldTime* to the length of the previous interval:

```
long OldTime, NewTime
NewTime = 1000 /* 1 second */

OldTime = GarbageCollectSetTimeLimit(NewTime)
```

See also

GarbageCollect
GarbageCollectGetTimeLimit

GetActiveSheet

Description
Returns the currently active sheet in an MDI frame window.

PocketBuilder	✗
PowerBuilder	✓

Applies to
MDI frame windows

Syntax
mdiframewindow.**GetActiveSheet** ( )

Return value
Window. Returns the sheet that is currently active in *mdiframewindow*. If no sheet is active, GetActiveSheet returns an invalid value. If *mdiframewindow* is null, GetActiveSheet returns null.

GetAlignment

Description
Obtains the alignment of the paragraph containing the insertion point in a RichTextEdit control.

PocketBuilder	✗
PowerBuilder	✓

Applies to
RichTextEdit controls

Syntax
rtename.**GetAlignment** ( )

Return value
Alignment. A value of the Alignment enumerated datatype indicating the alignment of the paragraph containing the insertion point.

GetAllowedImageAttributes

Description
Returns the set of allowed image attributes for a specific device.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✗

Applies to
Camera objects

Syntax

objectname.GetAllowedImageAttributes (*attrValue* })

Argument	Description
objectname	The name of the camera object that you want to inquire about
attrValue	An array of values of the CameraImageAttributes structure returned by reference that contains the set of attributes available on a specific device

Return value

Integer. Returns 1 for success, and one of the following negative values if an error occurs:

-1	Unspecified error
-2	Supporting DLL not loaded
-3	Other initialization error
-5	Inconsistency in this object instance
-6	Call to the driver or device failed
-7	Unsupported option
-8	Value for option is out of range

Usage

Use this function to determine which attributes, such as image size and zoom factors, can be set on the current device. For a list of values, see the CameraImageAttributes variable on the Enumerated tab page in the Browser.

Examples

This example gets the attributes that are available for a device in an array of CameraImageAttributes structures and displays them to the user so that the user can select the set of attributes to be used for preview and capture:

```
CameraImageAttributes AllowedConfigs[]
g_myCam.GetAllowedImageAttributes(AllowedConfigs)

// Display choices to user and let user select
// a preview and capture configuration
...
// User chose 1 for preview, 3 for capture
g_myCam.SetPreviewImageAttributes(AllowedConfigs[1])
g_myCam.SetCaptureImageAttributes(AllowedConfigs[3])
```

See also

CaptureImage
GetOption
Open
SetCaptureImageAttributes
SetPreviewImageAttributes

GetApplication

Description

Gets the handle of the current Application object so you can get and set properties of the application.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

GetApplication ( )

Return value

Application. Returns the handle of the current application object.

Usage

The GetApplication function lets you write generic code for an application, making it reusable in other applications. You do not have to code the actual name of the application when you want to set application properties.

Examples

To change whether Toolbar Tips are displayed, you can get the handle of the application object and set the ToolbarTips property:

```
application app
app = GetApplication()
app.ToolbarTips = FALSE
```

The previous example could be coded more simply as follows:

```
GetApplication().ToolbarTips = FALSE
```

GetAppointment

Description

Gets an appointment from Pocket Outlook.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✗

Applies to

POOM objects

Syntax

POOMAppointment *objectname*.GetAppointment (*index*)

Argument	Description
objectname	The name of the POOM object
index	Integer for the index of the appointment that you want to get

Return value

POOMAppointment. Use the IsValid function to confirm that a valid appointment was returned.

Usage

A user must be logged in to a POOM object to get an appointment from Pocket Outlook.

Examples

The following example logs in to the Pocket Outlook session and retrieves the first appointment in the list of appointments:

```
// global variable: g_poom
int li_return

g_poom = CREATE POOM
// log in to the Outlook session
li_return = g_poom.Login()

myAppt = g_poom.getAppointment( 1 )
if IsValid(myAppt) then
   // Use myAppt
end if
...
g_poom.Logout()
```

This example retrieves the first appointment and changes the subject and location:

```
integer li_rc
POOMAppointment appt

appt = g_poom.GetAppointment( 1 )
appt.Subject += " with Andre"
appt.Location = "Blue Room
li_rc = appt.update()

appt.display()
```

See also

Add
GetAppointmentFromOID
GetAppointments
Remove

GetAppointmentFromOID

Description — Gets an appointment from Pocket Outlook using the object ID.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✗

Applies to — POOM objects

Syntax — POOMAppointment *objectname*.GetAppointmentFromOID (*oid*)

Argument	Description
objectname	The name of the POOM object
oid	Unsignedlong for the object ID of the appointment that you want to get

Return value — POOMAppointment. Use the IsValid function to confirm that a valid appointment was returned.

Usage — A user must be logged in to a POOM object to get an appointment from Pocket Outlook.

Examples — The following example retrieves an appointment with an object ID of 1234:

```
// global variable: g_poom
POOMAppointment appt

appt = g_poom.GetAppointmentFromOID( 1234 )
If IsValid (appt) then
   // Use appt
end if
```

See also

Add
GetAppointment
GetAppointments
Remove

GetAppointments

Description

Gets an array of appointments from Pocket Outlook after optionally filtering the array for matching criteria.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✗

Applies to

POOM objects

Syntax

Integer *objectname*.GetAppointments ({*matchcriteria*,} *appointments* [ ])

Argument	Description
objectname	The name of the POOM object
matchcriteria	A string describing criteria you want to use to filter the list of appointments
appointments	An array of POOMAppointments passed by reference

Return value

Integer. Returns 1 for success and one of the following negative values if an error occurs:

-1 Unspecified error

-2 Cannot connect to the repository or a required internal subobject failed to connect to the repository

-3 Cannot log in to the repository

-4 Incorrect input argument

-5 Action cannot be performed

-6 The object identifier (OID) is not in the repository

-7 Feature is not implemented yet

-8 No matching entries found for the criteria

Usage

A user must be logged in to a POOM object to get appointments from Pocket Outlook.

Examples

The following example retrieves an array of appointments that satisfy the criterion that the location is the lunch room, and displays the array in a list box:

```
// global variable: g_poom
integer li_rc
POOMAppointment apptArray[]
POOMAppointment appt
```

```
String  sCriteria = "[Location] <> ~"Lunch Room~""
DateTime  dt
int li_idx

li_rc = g_poom.GetAppointments( sCriteria, apptArray )

FOR li_idx=1 to UPPERBOUND(apptArray)
   appt = apptArray[li_idx]
   lb_res.AddItem( "Appt(" + string(li_idx) + ")")
   lb_res.AddItem( "Subject: " + appt.Subject )
   lb_res.AddItem( "Location: " + appt.Location )
   dt = appt.appointmentstart
   lb_res.AddItem( "Start: " +   &
      string(dt, "dd-mmm-yyyy hh:mm") )
   lb_res.AddItem( "End: " +  &
      string(appt.appointmentEnd) )

   lb_res.AddItem( "Duration: " +   &
      string(appt.appointmentduration  ) )
   lb_res.AddItem( "Reminder: " +  &
      string(appt.reminderminutesbeforestart ) )

NEXT
```

See also

Add
GetAppointment
GetAppointmentFromOID
Remove

GetArgElement

Description

Returns the value in the specified argument.

PocketBuilder	✗
PowerBuilder	✓

Applies to

Window ActiveX controls

Syntax

activexcontrol.**GetArgElement** (*index*)

Return value

Any. Returns the specified argument.

GetAsBitmap

Description

Converts the current image of an object derived from the GraphicObject baseclass to a standard Windows bitmap.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✗

Syntax

String GetAsBitmap (*graphicObject* , *bitmap*)

Argument	Description
graphicObject	Read-only value for current object that inherits from GraphicObject
bitmap	Blob variable passed by reference for the bitmap created from a GraphicObject descendent

Return value

Integer. Returns 1 for success and a negative number if an error occurs.

Usage

You assign the visual object that you want to capture in the first argument to the GetAsBitmap fucntion, and reference the bitmap you want to create as a Blob datatype in the second argument to the function. The bitmap can then be saved to a file or set in a picture control.

Examples

This example in a command button Clicked event saves the button image in a Picture control:

```
int  li_ret
BLOB lb_bmp

li_ret = GetAsBitmap (this, lb_bmp)
p_result.SetPicture(lb_bmp)
```

GetAutomationNativePointer

Description

Gets a pointer to the OLE object associated with the OLEObject variable. The pointer lets you call OLE functions in an external DLL for the object.

PocketBuilder	✗
PowerBuilder	✓

Applies to

OLEObject

Syntax

oleobject.**GetAutomationNativePointer** (*pointer*)

Return value Integer. Returns 0 if it succeeds and -1 if an error occurs.

GetCertificateLabel

Description Called by EAServer to allow the user to select one of the available SSL certificate labels for authentication. This function is used by PowerBuilder clients connecting to EAServer.

PocketBuilder	✗
PowerBuilder	✓

Applies to SSLCallBack objects

Syntax *sslcallback*.**GetCertificateLabel** (*thesessioninfo, labels*)

Return value String. Returns one of the labels passed to the function.

GetChildrenList

Description Provides a list of the children of a routine included in a trace tree model.

PocketBuilder on Desktop	✓
PocketBuilder on Pocket PC	✗
PocketBuilder on Smartphone	✗
PowerBuilder	✓

Applies to TraceTreeObject, TraceTreeRoutine, and TraceTreeGarbageCollect objects

Syntax *instancename*.**GetChildrenList** (*list*)

Argument	Description
instancename	Instance name of the TraceTreeObject, TraceTreeRoutine, or TraceTreeGarbageCollect object.
list	An unbounded array variable of datatype TraceTreeNode in which GetChildrenList stores a TraceTreeNode object for each child of a routine. This argument is passed by reference.

Return value

ErrorReturn. Returns the following values:

- Success!—The function succeeded
- ModelNotExistsError!—The model does not exist

Usage

You use the GetChildrenList function to extract a list of the children of a routine (the classes and routines it calls) included in a trace tree model. Each child listed is defined as a TraceTreeNode object and provides the type of activity represented by that child.

You must have previously created the trace tree model from a trace file using the BuildModel function.

When the GetChildrenList function is called for TraceTreeGarbageCollect objects, each child listed usually represents the destruction of a garbage collected object.

Examples

This example checks the activity type of a node included in the trace tree model. If the activity type is an occurrence of a routine, it determines the name of the class that contains the routine and provides a list of the classes and routines called by that routine:

```
TraceTree ltct_node
TraceTreeNode ltctn_list
...
CHOOSE CASE node.ActivityType
    CASE ActRoutine!
    TraceTreeRoutine ltctrt_rout
    ltctrt_rout = ltct_node
    result += "Enter " + ltctrt_rout.ClassName &
    + "." + ltctrt_rout.name + " " &
    + String(ltctrt_rout.ObjectID) + " " &
    + String(ltctrt_rout.EnterTimerValue) &
     + "~r~n" ltctrt_rout.GetChildrenList(ltctn_list)
...
```

See also

BuildModel

GetColumn

Description

Retrieves column information for a DataWindow, child DataWindow, or ListView control.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

For syntax for a DataWindow or a child DataWindow, see the GetColumn method for DataWindows in the *DataWindow Reference* or the online Help.

Applies to

ListView controls

Syntax

listviewname.**GetColumn** (*index, label, alignment, width*)

Argument	Description
listviewname	The name of the ListView control from which you want to find the properties for a column.
index	An integer whose value is the index of the column for which you want to find properties.
label	A string identifying the label of the column for which you want to find properties. This argument is passed by reference.
alignment	A value of the enumerated datatype Alignment specifying the alignment of the column for which you want to find properties. Values are: • Center! • Justify! • Left! • Right! This argument is passed by reference.
width	An integer whose value is the width of the column for which you want to find properties. This argument is passed by reference.

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs.

Usage

Use *label*, *alignment*, and *width* to retrieve the properties for a specified column.

Examples

This example uses the instance variable *li_col* to pass the column number to GetColumn and retrieve the properties for the column. The script uses SetColumn to change the column's alignment:

```
string ls_label,ls_align
int li_width
```

```
alignment la_align

IF lv_list.View <> ListViewReport! THEN
    lv_list.View = ListViewReport!
END IF

IF li_col = 0 THEN
    MessageBox("Error!","Click on a Column bar.", &
      StopSign!)
ELSE
    lv_list.GetColumn(li_col, ls_label, la_align, &
      li_width)
    lv_list.SetColumn(li_col, ls_label, Right!, &
      li_width)
END IF
```

See also SetColumn

GetCommandDDE

Description Obtains the command sent by the client application when your application is a DDE server.

PocketBuilder	✗
PowerBuilder	✓

Syntax **GetCommandDDE** (*string*)

Return value Integer. Returns 1 if it succeeds and -1 if an error occurs (such as the function was called in the wrong context). If *string* is null, GetCommandDDE returns null.

GetCommandDDEOrigin

Description When called by the DDE server application, obtains the application name parameter used by the DDE client sending the command.

PocketBuilder	✗
PowerBuilder	✓

Syntax **GetCommandDDEOrigin** (*applstring*)

Return value Integer. Returns 1 if it succeeds and -1 if an error occurs (such as the function was called in the wrong context). If *applstring* is null, GetCommandDDEOrigin returns null.

GetCommandString

Description Reserved for future use. Returns the command string sent by dbmlsync to the synchronization server.

Applies to MLSync controls

Syntax *syncObject*.**GetCommandString** ( )

Argument	Description
syncObject	The name of the MLSync object that starts a synchronization for which you want to get the actual dbmlsync command submitted to the synchronization server.

Return value String. Returns the command string that is set for submission to the synchronization server.

GetCompanyName

Description Returns the company name for the current execution context.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to ContextInformation objects

Syntax *servicereference*.**GetCompanyName** (*name*)

Argument	Description
servicereference	Reference to the ContextInformation service instance.
name	String into which the function places the company name. This argument is passed by reference.

Return value

Integer. Returns 1 if the function succeeds and -1 if an error occurs.

Usage

Call this function to determine the company name (such as Sybase, Inc.).

Examples

This example calls the GetCompanyName function:

```
String ls_company
Integer li_return
ContextInformation ci

ci = create ContextInformation
//or GetContextService("ContextInformation", ci)
li_return = ci.GetCompanyName(ls_company)
IF li_return = 1 THEN
    sle_co_name.text = ls_company
END IF
```

See also

GetContextService
GetFixesVersion
GetHostObject
GetMajorVersion
GetMinorVersion
GetName
GetShortName
GetVersionName

GetContact

Description

Gets a contact from Pocket Outlook.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✗

Applies to

POOM objects

Syntax

POOMContact *objectname*.GetContact (*index*)

Argument	Description
objectname	The name of the POOM object
index	Integer for the index of the contact that you want to get

Return value
POOMContact. Use the IsValid function to confirm that a valid contact was returned.

Usage
A user must be logged in to a POOM object to get a contact from Pocket Outlook.

Examples
The following example retrieves the first contact in Pocket Outlook:

```
// global variable: g_poom
POOMContact myContact
myContact = g_poom.getContact( 1 )

if IsValid(myContact) then
   // Use myContact
end if
```

See also
Add
GetContactFromOID
GetContacts
Remove

GetContactFromOID

Description
Gets a contact from Pocket Outlook using the object ID.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✗

Applies to
POOM objects

Syntax
POOMContact *objectname*.GetContactFromOID (*oid*)

Argument	Description
objectname	The name of the POOM object
oid	Unsignedlong for the object ID of the contact that you want to get

Return value
POOMContact. Use the IsValid function to confirm that a valid contact was returned.

Usage
A user must be logged in to a POOM object to get a contact from Pocket Outlook.

Examples
The following example retrieves a contact with an object ID of 321:

```
myContact = g_poom.getContactFromOID( 321 )
if IsValid(myContact) then
   // Use myContact
end if
```

See also

Add
GetContact
GetContacts
Remove

GetContacts

Description

Gets an array of contacts from Pocket Outlook after optionally filtering the array for matching criteria.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✗

Applies to

POOM objects

Syntax

Integer *objectname*.GetContacts ({*matchcriteria*,} *contacts* [ ])

Argument	Description
objectname	The name of the POOM object
matchcriteria	A string describing criteria you want to use to filter the list of contacts
contacts	An array of POOMContacts passed by reference

Return value

Integer. Returns 1 for success and one of the following negative values if an error occurs:

-1 Unspecified error

-2 Cannot connect to the repository or a required internal subobject failed to connect to the repository

-3 Cannot log in to the repository

-4 Incorrect input argument

-5 Action cannot be performed

-6 The object identifier (OID) is not in the repository

-7	Feature is not implemented yet
-8	No matching entries found for the criteria

Usage

A user must be logged in to a POOM object to get contacts from Pocket Outlook.

Examples

The following example retrieves contacts that match the criterion that the contact’s Department property is Finance, and writes their names, phone numbers, and e-mail addresses to a list box:

```
// global variable: g_poom
integer li_rc
POOMContact contactArray[]
POOMContact contact
String  sCriteria = "[Department] = ~"Finance~""
DateTime  dt
int idx

li_rc = g_poom.GetContacts( sCriteria, contactArray )
lb_res.AddItem( "Contact[] ret: " + string(li_rc) )

FOR idx=1 to UPPERBOUND(contactArray)
   contact = contactArray[idx]
   lb_res.AddItem( "Contact(" + string(idx) + ")")
   lb_res.AddItem( "First: " + contact.FirstName )
   lb_res.AddItem( "Last: " + contact.LastName )
   lb_res.AddItem( "Phone1: " + &
      contact.businesstelephonenumber )
   lb_res.AddItem("E-Mai1 1: " + contact.email1address)
NEXT

lb_res.SelectItem( lb_res.TotalItems() )
```

See also

Add
GetContact
GetContactFromOID
Remove

GetContextKeywords

Description

Retrieves one or more values associated with a specified keyword.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

ContextKeyword objects

Syntax

servicereference.**GetContextKeywords** (*name*, *values*)

Argument	Description
servicereference	Reference to the ContextKeyword service instance.
name	String specifying the keyword for which the function returns corresponding values.
values	Unbounded String array into which the function places the values that correspond to *name*. This argument is passed by reference.

Return value

Integer. Returns the number of elements in *values* if the function succeeds and -1 if an error occurs.

Usage

Call this function to access environment variables. Environment-variable availability differs by execution context:

- **PocketBuilder design time** The function accesses DOS environment variables, each of which has a unique keyword.
- **PocketBuilder runtime** Retrieves a blank string since there are no DOS environment variables to access on Pocket PC devices.

Examples

This example calls the GetContextKeywords function:

```
String ls_keyword
Integer li_count, li_return
ContextKeyword lcx_key

li_return = this.GetContextService &
    ("ContextKeyword", lcx_key)
ls_keyword = sle_name.Text
lcx_key.GetContextKeywords &
    (ls_keyword, is_values)
FOR li_count = 1 to UpperBound(is_values)
    lb_parms.AddItem(is_values[li_count])
NEXT
```

See also

GetContextService

GetContextService

Description

Creates a reference to a context-specific instance of the specified service.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Any object

Syntax

GetContextService (*servicename*, *servicereference*)

Argument	**Description**
servicename	String specifying the service object. Valid values are: • ContextInformation—Context information service • ContextKeyword—Context keyword service • CORBACurrent—(PowerBuilder only) CORBA current service for client- or component-management of EAServer transactions • ErrorLogging—(PowerBuilder only) Error logging service for PowerBuilder components running in a transaction server such as EAServer or COM+ • Internet—Internet service • SSLServiceProvider—(PowerBuilder only) SSL service provider service that allows PowerBuilder clients to establish SSL connections to EAServer components • TransactionServer—(PowerBuilder only) Transaction server service for PowerBuilder components running in a transaction server such as EAServer or COM+
servicereference	PowerObject into which the function places a reference to the service object specified by *servicename*. This argument is passed by reference.

Return value

Integer. Returns 1 if the function succeeds and a negative integer if an error occurs. The return value -1 indicates an unspecified error.

Usage

Call this function to establish a reference to a service object, allowing you to access methods and properties in the service object. You must call this function before calling service object functions.

Using a CREATE statement
You can instantiate these objects with a PowerScript CREATE statement. However, this always creates an object for the default context (native PocketBuilder execution environment), regardless of where the application is running.

Examples

This example calls the GetContextService function and displays the class of the service in a single line edit box:

```
Integer li_return
ContextKeyword lcx_key

li_return = this.GetContextService &
    ("ContextKeyword", lcx_key)
sle_classname.Text = ClassName(lcx_key)
...
```

See also

BeginTransaction
GetCompanyName
GetContextKeywords
GetHostObject
GetMajorVersion
GetMinorVersion
GetName
GetShortName
GetURL
GetVersionName
HyperLinkToURL
Init
PostURL

GetCredentialAttribute

Description

Called by EAServer to allow the user to supply user credentials dynamically. This function is used by PowerBuilder clients connecting to EAServer.

PocketBuilder	✗
PowerBuilder	✓

Applies to

SSLCallBack objects

Syntax *sslcallback*.**GetCredentialAttribute** (*thesessioninfo, attr, attrvalues*)

Return value String. Returns the selected attribute value.

GetCurrentDirectory

Description Gets the current directory for your target application.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax **GetCurrentDirectory** ( )

Return value String. Returns the full path name for the current directory.

Examples This example puts the current directory name in a SingleLineEdit control:

```
sle_1.text = GetCurrentDirectory( )
```

See also
ChangeDirectory
CreateDirectory
DirectoryExists
RemoveDirectory

GetData

Obtains data from a control.

To obtain	Use
The value of a data point in a series in a graph	Syntax 1
The unformatted data from an EditMask control	Syntax 2
Data from an OLE server	Syntax 3

Syntax 1 For data points in graphs

Description

Gets the value of a data point in a series in a graph.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Graph controls in windows and user objects, and graphs in DataWindow controls

Syntax

controlname.**GetData** ({ *graphcontrol*, } *seriesnumber*, *datapoint* {, *datatype* })

Argument	Description
controlname	The name of the graph from which you want data, or the name of the DataWindow control containing the graph.
graphcontrol (DataWindow control only)	(Optional) A string whose value is the name of the graph from which you want the data when *controlname* is a DataWindow.
seriesnumber	The number that identifies the series from which you want data.
datapoint	The number of the data point for which you want the value.
datatype (scatter graph only)	(Optional) A value of the grDataType enumerated datatype specifying whether you want the x or y value of the data point in a scatter graph. Values are: • xValue! — The x value of the data point • yValue! — (Default) The y value of the data point

Return value

Double. Returns the value of the data in *datapoint* if it succeeds and 0 if an error occurs. If any argument's value is null, GetData returns null.

Usage

You can use GetData only for graphs whose values axis is numeric. For graphs with other types of values axes, use the GetDataValue function instead.

Examples

These statements obtain the data value of data point 3 in the series named Costs in the graph gr_computers in the DataWindow control dw_equipment:

```
integer SeriesNbr
double data_value

// Get the number of the series.
SeriesNbr = &
    dw_equipment.FindSeries("gr_computers", "Costs")
data_value = dw_equipment.GetData( &
    "gr_computers" , SeriesNbr, 3)
```

These statements obtain the data value of the data point under the mouse pointer in the graph gr_prod_data and store it in *data_value*:

```
integer SeriesNbr, ItemNbr
double data_value
grObjectType MouseHit

MouseHit = &
    gr_prod_data.ObjectAtPointer(SeriesNbr, ItemNbr)
IF MouseHit = TypeSeries! THEN
    data_value = &
      gr_prod_data.GetData(SeriesNbr, ItemNbr)
END IF
```

These statements obtain the x value of the data point in the scatter graph gr_sales_yr and store it in *data_value*:

```
integer SeriesNbr, ItemNbr
double data_value

gr_product_data.ObjectAtPointer(SeriesNbr, ItemNbr)
data_value = &
    gr_sales_yr.GetData(SeriesNbr, ItemNbr, xValue!)
```

See also

DeleteData
FindSeries
GetDataValue
InsertData
ObjectAtPointer

Syntax 2 For EditMask controls

Description

Gets the unformatted text from an EditMask control.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

EditMask controls

Syntax

editmaskname.**GetData** (*datavariable*)

Argument	Description
editmaskname	The name of the EditMask control containing the data.

Argument	Description
datavariable	A variable to which GetData will assign the unformatted data in the EditMask control. The datatype of *datavariable* must match the datatype of the EditMask control, which you select in the Window painter. Available datatypes are date, DateTime, decimal, double, string, and time.

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is null, GetData returns null.

Usage

You can find out the datatype of an EditMask control by looking at its MaskDataType property, which holds a value of the MaskDataType enumerated datatype.

Examples

This example gets data of datatype date from the EditMask control em_date. Formatting characters for the date are ignored. The String function converts the date to a string so it can be assigned to the SingleLineEdit sle_date:

```
date d
em_date.GetData(d)
sle_date.Text = String(d, "mm-dd-yy")
```

This example gets string data from the EditMask control em_string and assigns the result to sle_string. Characters in the edit mask are ignored:

```
string s
em_string.GetData(s)
sle_string.Text = s
```

Syntax 3 For data in an OLE server

Description

Gets data from the OLE server associated with an OLE control using Uniform Data Transfer.

PocketBuilder	✗
PowerBuilder	✓

Applies to

OLE controls and OLE custom controls

Syntax

olename.**GetData** (*clipboardformat*, *data*)

Return value

Integer. Returns 0 if it succeeds and -1 if an error occurs.

GetDataAsBitmap

Description

Retrieves the data in the control as a standard Windows bitmap that is compatible with the Picture control and Windows desktop applications.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✗

Applies to

Signature controls

Syntax

Integer *controlname*.GetDataAsBitmap (*data*)

Argument	Description
controlname	The name of the control for which you want to retrieve the data
data	The blob in which the bitmap data is saved

Return value

Integer. Returns 1 for success and a negative integer for failure.

Usage

The GetDataAsBitmap function returns both typed and freehand drawing or writing from a signature control as a bitmap in a blob.

Examples

The following example gets unformatted data from a Signature control as a bitmap and writes it to a file. It also displays the bitmap in a Picture control:

```
blob lblb_bmp
integer li_file, li_rtn

li_rtn = sig_1.GetDataAsBitMap(lblb_bmp)
sle_1.text = string(li_rtn)

li_file = FileOpen("\My Documents\testi.bmp", &
   StreamMode!, Write!, LockWrite!, Replace!)

FileWrite( li_file, lblb_bmp )
FileClose( li_file )

p_1.setpicture(lblb_bmp)
```

See also

GetDataAsInk
GetDataAsRTF
GetDataAsText

GetDataAsInk

Description

Retrieves the data in the control in Pocket Word Ink (PWI) format. This format is compatible with Pocket Word.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✗

Applies to

Signature controls

Syntax

Integer *controlname*.GetDataAsInk (*data*)

Argument	Description
controlname	The name of the control for which you want to retrieve the data
data	The blob in which the PWI data is saved

Return value

Integer. Returns 1 for success and a negative integer for failure.

Usage

The GetDataAsInk function returns both typed and freehand drawing or writing from a Signature control in PWI format in a blob.

Examples

The following example gets unformatted data from a Signature control in PWI format and writes it to a file. The return value from the function is written to a single line edit box:

```
blob lblb_ink
integer li_file, li_rtn

li_rtn = sig_1.GetDataAsInk(lblb_ink)
sle_1.text = string(li_rtn)

li_file = FileOpen("\My Documents\testpwi.pwi", &
   StreamMode!, Write!, LockWrite!, Replace!)
FileWrite( li_file, lblb_ink )
FileClose( li_file )
```

See also

GetDataAsBitmap
GetDataAsRTF
GetDataAsText
SetDataAsInk

GetDataAsRTF

Description

Retrieves the text data in the control as an RTF ANSI text block in a blob or Unicode string. This function does not return graphic data.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✗

Applies to

Signature controls

Syntax

Integer *controlname*.GetDataAsRTF (*data*)

Argument	Description
controlname	The name of the control for which you want to retrieve the data
data	The blob or string in which the RTF data is saved

Return value

Integer. Returns 1 for success and a negative integer for failure.

Usage

The GetDataAsRTF function returns text data from a Signature control in PWI format in a blob or a Unicode string. Only text data, such as text entered using the SIP, is retrieved.

Examples

The following example gets text data from a Signature control in RTF format as a blob and as a string and writes the RTF to two separate files:

```
blob lblb_rtf
string ls_rtf
integer li_file, li_rtn

// Get data as a blob
li_rtn = sig_1.GetDataAsRTF(lblb_rtf)
sle_1.text = string(li_rtn)

li_file = FileOpen("\My Documents\blob.rtf",   &
   StreamMode!, Write!, LockWrite!, Replace!)
FileWrite( li_file, lblb_rtf )
FileClose( li_file )

// Get data as a string
li_rtn = sig_1.GetDataAsRTF(ls_rtf)
sle_1.text += ", " + String(li_rtn)

li_file = FileOpen("\My Documents\string.rtf",  &
   StreamMode!, Write!, LockWrite!, Replace!)
FileWrite( li_file, ls_rtf )
FileClose( li_file )
```

See also
GetDataAsBitmap
GetDataAsInk
GetDataAsText
SetDataAsRTF

GetDataAsText

Description
Retrieves the text data in the control as a string. This function returns data typed into the control using the SIP. It does not return graphic signature data.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✗

Applies to
Signature controls

Syntax
Integer *controlname*.GetDataAsText (*data*)

Argument	Description
controlname	The name of the control for which you want to retrieve the data
data	The string in which the text data is saved

Return value
Integer. Returns 1 for success and a negative integer for failure.

Usage
The GetDataAsText function returns text data from a Signature control in PWI format in a Unicode string. Only text data, such as text entered using the SIP, is retrieved.

Examples
The following example gets text data from a Signature control as a Unicode string:

```
string ls_text
integer li_rtn

li_rtn = sig_1.GetDataAsText(ls_text)
sle_1.text = String(li_rtn)
```

See also
GetDataAsBitmap
GetDataAsInk
GetDataAsRTF
SetDataAsText

GetDataDDE

Description

Obtains data sent from another DDE application and stores it in the specified string variable. PowerBuilder can use GetDataDDE when acting as a DDE client or a DDE server application.

PocketBuilder	✗
PowerBuilder	✓

Syntax

GetDataDDE (*string*)

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs (such as the function was called in the wrong context). If *string* is null, GetDataDDE returns null.

GetDataDDEOrigin

Description

Determines the origin of data from a hot-linked DDE server application or a DDE client application, and if successful, stores the application's DDE identifiers in the specified strings. PowerBuilder can use GetDataDDEOrigin when it is acting as a DDE client or as a DDE server application.

PocketBuilder	✗
PowerBuilder	✓

Syntax

GetDataDDEOrigin (*applstring*, *topicstring*, *itemstring*)

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs (such as the function was called in the wrong context). If any argument's value is null, GetDataDDEOrigin returns null.

GetDataPieExplode

Description

Reports the percentage of the pie graph's radius that a pie slice is exploded. An exploded slice is moved away from the center of the pie in order to draw attention to the data.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Graph controls in windows and user objects, and graphs in DataWindow controls

Syntax

controlname.**GetDataPieExplode** ({ *graphcontrol*, } *series*, *datapoint*, *percentage*)

Argument	Description
controlname	The name of the graph for which you want the percentage a pie slice is exploded, or the name of the DataWindow control containing the graph
graphcontrol (DataWindow control only)	(Optional) A string whose value is the name of the graph in the DataWindow control for which you want the percentage a pie slice is exploded
series	The number that identifies the series
datapoint	The number of the exploded data point (that is, the pie slice)
percentage	An integer variable in which you want to store the percentage of the graph's radius that the pie slice is exploded

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is null, GetDataPieExplode returns null.

Examples

This example reports the percentage that a pie slice is exploded when the user clicks on that slice. The code checks whether the graph is a pie graph using the property Graphite. It then finds out whether the user clicked on a pie slice by checking the series and data point values set by ObjectAtPointer. The script is for the DoubleClicked event of a graph object:

```
integer series, datapoint
grObjectType clickedtype
integer percentage

percentage = 50
IF (This.GraphType <> PieGraph! and &
    This.GraphType <> Pie3D!) THEN RETURN
clickedtype = This.ObjectAtPointer(series, &
    datapoint)

IF (series > 0 and datapoint > 0) THEN
    This.GetDataPieExplode(series, datapoint, &
    percentage)
    MessageBox("Explosion Percentage", &
      "Data point " + This.CategoryName(datapoint) &
      + " in series " + This.SeriesName(series) &
      + " is exploded " + String(percentage) + "%")
END IF
```

See also

SetDataPieExplode

GetDataStyle

Finds out the appearance of a data point in a graph. Each data point in a series can have individual appearance settings. There are different syntaxes, depending on what settings you want to check.

To get the	Use
Data point's colors	Syntax 1
Line style and width used by the data point	Syntax 2
Fill pattern or symbol for the data point	Syntax 3

GetDataStyle provides information about a single data point. The series to which the data point belongs has its own style settings. In general, the style values for the data point are the same as its series' settings. Use SetDataStyle to change the style values for individual data points. Use GetSeriesStyle and SetSeriesStyle to get and set style information for the series.

The graph stores style information for properties that do not apply to the current graph type. For example, you can find out the fill pattern for a data point or a series in a 2-dimensional line graph, but that fill pattern will not be visible.

For the enumerated datatype values that GetDataStyle stores in *linestyle* and *enumvariable,* see SetDataStyle.

Syntax 1 For the colors of a data point

Description

Obtains the colors associated with a data point in a graph.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Graph controls in windows and user objects, and graphs in DataWindow controls

Syntax

controlname.**GetDataStyle** ({ *graphcontrol*, } *seriesnumber*, *datapointnumber*, *colortype*, *colorvariable*)

Argument	Description
controlname	The name of the graph for which you want the color of a data point, or the name of the DataWindow control containing the graph.

Argument	Description
graphcontrol (Data Window control only)	(Optional) When *controlname* is a DataWindow control, the name of the graph for which you want the color of a data point.
seriesnumber	The number of the series in which you want the color of a data point.
datapointnumber	The number of the data point for which you want the color.
colortype	A value of the grColorType enumerated datatype specifying the aspect of the data point for which you want the color. Values are: • Background! — The background color • Foreground! — Text (fill color) • LineColor! — The color of the line • Shade! — The shaded area of three-dimensional graphics
colorvariable	A long variable in which you want to store the color.

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs. Stores a color value in *colorvariable*. If any argument's value is null, GetDataStyle returns null.

Examples

This example gets the text (foreground) color used for data point 6 in the series named Salary in the graph gr_emp_data. It stores the color value in the variable *color_nbr*:

```
long color_nbr
integer SeriesNbr

// Get the number of the series
SeriesNbr = gr_emp_data.FindSeries("Salary")

// Get the color
gr_emp_data.GetDataStyle(SeriesNbr, 6, &
    Foreground!, color_nbr)
```

This example gets the background color used for data point 6 in the series entered in the SingleLineEdit sle_series in the DataWindow graph gr_emp_data. It stores the color value in the variable *color_nbr*:

```
long color_nbr
integer SeriesNbr

// Get the number of the series
SeriesNbr = FindSeries("gr_emp_data", sle_series.Text)


// Get the color
```

```
dw_emp_data.GetDataStyle("gr_emp_data", &
    SeriesNbr, 6, Background!, color_nbr)
```

See also

FindSeries
GetSeriesStyle
SetDataStyle
SetSeriesStyle

Syntax 2 For the line style and width used by a data point

Description

Obtains the line style and width for a data point in a graph.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Graph controls in windows and user objects, and graphs in DataWindow controls

Syntax

controlname.**GetDataStyle** ({ *graphcontrol*, } *seriesnumber*, *datapointnumber*, *linestyle*, *linewidth*)

Argument	Description
controlname	The name of the graph for which you want the line style and width of a data point, or the name of the DataWindow control containing the graph.
graphcontrol (DataWindow control only)	(Optional) A string whose value is the name of the graph (in the DataWindow control) for which you want the line style and width of a data point.
seriesnumber	The number of the series in which you want the line style and width of a data point.
datapointnumber	The number of the data point for which you want the line style and width.
linestyle	A variable of type LineStyle in which you want to store the line style.
linewidth	An integer variable in which you want to store the width of the line. The width is measured in pixels.

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs. For the specified series and data point, stores its line style in *linestyle* and the line's width in *linewidth*. If any argument's value is null, GetDataStyle returns null.

Usage

For the enumerated datatype values that GetDataStyle will store in *linestyle*, see SetDataStyle.

Examples

This example gets the line style and width of data point 10 in the series named Costs in the graph gr_product_data. It stores the information in the variables *line_style* and *line_width*:

```
integer SeriesNbr, line_width
LineStyle line_style

// Get the number of the series
SeriesNbr = gr_product_data.FindSeries("Costs")
gr_product_data.GetDataStyle(SeriesNbr, 10, &
    line_style, line_width)
```

This example gets the line style and width for data point 6 in the series entered in the SingleLineEdit sle_series in the graph gr_depts in the DataWindow control dw_employees. The information is stored in the variables *line_style* and *line_width*:

```
integer SeriesNbr, line_width
LineStyle line_style

// Get the number of the series
SeriesNbr = dw_employees.FindSeries( &
   " gr_depts " , sle_series.Text)

// Get the line style and width
dw_employees.GetDataStyle("gr_depts", SeriesNbr, &
    6, line_style, line_width)
```

See also

FindSeries
GetSeriesStyle
SetDataStyle
SetSeriesStyle

Syntax 3 For the fill pattern or symbol of a data point

Description

Obtains the fill pattern or symbol of a data point in a graph.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Graph controls in windows and user objects, and graphs in DataWindow controls

Syntax

controlname.**GetDataStyle** ({ *graphcontrol*, } *seriesnumber*, *datapointnumber*, *enumvariable*)

Argument	Description
controlname	The name of the graph for which you want the fill pattern or symbol type of a data point, or the name of the DataWindow control containing the graph.
graphcontrol (DataWindow control only) (optional)	A string whose value is the name of the graph (in the DataWindow control) for which you want the fill pattern or symbol type of a data point.
seriesnumber	The number of the series in which you want the fill pattern or symbol type of a data point.
datapointnumber	The number of the data point for which you want the fill pattern or symbol type.
enumvariable	The variable in which you want to store the data style. You can specify a FillPattern or grSymbolType variable. The data style information stored will depend on the variable type.

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs. Stores, according to the type of *enumvariable*, a value of that enumerated datatype representing the fill pattern or symbol used for the specified data point. If any argument's value is null, GetDataStyle returns null.

Usage

For the enumerated datatype values that GetDataStyle will store in *enumvariable*, see SetDataStyle.

Examples

This example gets the pattern used to fill data point 10 in the series named Costs in the graph gr_product_data. The information is stored in the variable *data_pattern*:

```
integer SeriesNbr
FillPattern data_pattern

// Get the number of the series
SeriesNbr = gr_product_data.FindSeries("Costs")
gr_product_data.GetDataStyle(SeriesNbr, 10, &
    data_pattern)
```

This example gets the pattern used to fill data point 6 in the series entered in the SingleLineEdit sle_series in the graph gr_depts in the DataWindow control dw_employees. The information is assigned to the variable *data_pattern*:

```
integer SeriesNbr
FillPattern data_pattern

// Get the number of the series
SeriesNbr = dw_employees.FindSeries("gr_depts", &
    sle_series.Text)

// Get the pattern
dw_employees.GetDataStyle("gr_depts", SeriesNbr, &
    6, data_pattern)
```

These statements store in the variable symbol_type the symbol of data point 10 in the series named Costs in the graph gr_product_data:

```
integer SeriesNbr
grSymbolType symbol_type

// Get the number of the series
SeriesNbr = gr_product_data.FindSeries("Costs")
gr_product_data.GetDataStyle(SeriesNbr, 10, &
    symbol_type)
```

These statements store the symbol for a data point in the variable *symbol_type*. The data point is the sixth point in the series named in the SingleLineEdit sle_series in the graph gr_depts in the DataWindow control dw_employees:

```
integer SeriesNbr
grSymbolType symbol_type

// Get the number of the series
SeriesNbr = dw_employees.FindSeries("gr_depts", &
    sle_series.Text)

// Get the symbol
dw_employees.GetDataStyle("gr_depts", SeriesNbr, &
    6, symbol_type)
```

See also

FindSeries
GetSeriesStyle
SetDataStyle
SetSeriesStyle

GetDataValue

Description

Obtains the value of a data point in a series in a graph.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Graph controls in windows and user objects, and graphs in DataWindow controls

Syntax

controlname.**GetDataValue** ({ *graphcontrol*, } *seriesnumber*, *datapoint*, *datavariable* {, *xory* })

Argument	Description
controlname	The name of the graph from which you want data, or the name of the DataWindow control containing the graph.
graphcontrol (DataWindow control only)	(Optional) A string whose value is the name of the graph in the DataWindow control from which you want the data.
seriesnumber	The number that identifies the series from which you want data.
datapoint	The number of the data point for which you want the value.
datavariable	The name of a variable that will hold the data value. The variable's datatype can be date, DateTime, double, string, or time. The variable must have the same datatype as the values axis of the graph.
xory (scatter graph only)	(Optional) A value of the grDataType enumerated datatype specifying whether you want the x or y value of the data point in a scatter graph. Values are: • xValue! — The x value of the data point • yValue! — (Default) The y value of the data point

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is null, GetDataValue returns null.

Usage

GetDataValue retrieves data from any graph. The data is stored in *datavariable*, whose datatype must match the datatype of the graph's values axis. If the values axis is numeric, you can also use the GetData function.

Examples

These statements obtain the data value of data point 3 in the series named Costs in the graph gr_computers in the DataWindow control dw_equipment:

```
integer SeriesNbr, rtn
double data_value
```

```
// Get the number of the series.
SeriesNbr = dw_equipment.FindSeries( &
    "gr_computers", "Costs")
rtn = dw_equipment.GetDataValue( &
    "gr_computers" , SeriesNbr, 3, data_value)
```

These statements obtain the data value of the data point under the mouse pointer in the graph gr_prod_data and store it in *data_value*. If the user does not click on a data point, then *ItemNbr* is set to 0. The categories of the graph are time values:

```
integer SeriesNbr, ItemNbr, rtn
time data_value
grObjectType MouseHit

MouseHit = &
    gr_prod_data.ObjectAtPointer(SeriesNbr, ItemNbr)
IF ItemNbr > 0 THEN
    rtn = gr_prod_data.GetDataValue( &
      SeriesNbr, ItemNbr, data_value)
END IF
```

These statements obtain the x value of the data point in the scatter graph gr_sales_yr and store it in *data_value*. If the user does not click on a data point, then *ItemNbr* is set to 0. The datatype of the category axis is Date:

```
integer SeriesNbr, ItemNbr, rtn
date data_value

gr_product_data.ObjectAtPointer(SeriesNbr, ItemNbr)
IF ItemNbr > 0 THEN
    rtn = gr_sales_yr.GetDataValue( &
      SeriesNbr, ItemNbr, data_value, xValue!)
END IF
```

See also

DeleteData
FindSeries
InsertData
ObjectAtPointer

GetDbmlsyncPath

Description

Reserved for future use. Retrieves the full path and file name of the *dbmlsync.exe* that is installed on the workstation.

Applies to

MLSync and MLSynchronization controls

Syntax

SyncObject.**GetDbmlsyncPath** ( )

Argument	Description
syncObject	The name of the synchronization object

Return value

String. Returns the value of full path and file name of the synchronization executable. Returns -1 if.*dbmlsync.exe* is not found.

GetDeskRect

Description

On a Pocket PC device or emulator, gets the rectangular coordinates, in pixels, of the current window—which does not include the area occupied by the Soft Input Panel (SIP) when the latter is visible. On the desktop, GetDeskRect gets the coordinates of the desktop monitor.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✗

Syntax

int GetDeskRect (long *left*, long *top*, long *right*, long *bottom*)

Argument	Description
left	Leftmost value (typically 0) of the rectangular area occupied by the: • current window on a Pocket PC device or emulator • monitor for the desktop
top	Topmost value of the rectangular area occupied by the: • current window on a Pocket PC device or emulator (typically 26) • monitor for the desktop (typically 0)

Argument	Description
right	Rightmost value of the rectangular area occupied by the: • current window on a Pocket PC device or emulator (typically 240) • monitor for the desktop (for example, 1024 on a monitor with a desktop area of 1024 by 768 pixels)
bottom	Bottommost value of the rectangular area occupied by: • current window on a Pocket PC device or emulator (typically 214 when the SIP is visible and 320 when the SIP is not visible) • monitor for the desktop (for example, 768 on a monitor with a desktop area of 1024 by 768 pixels)

Return value

Integer. Returns 1 for success and -1 for failure.

Usage

Typically it is useful to know the bottommost coordinate of the current window when the SIP is visible. That way you can adjust the positions and sizes of controls on the window to account for the smaller display size of the window when the SIP is being used.

Examples

The following example displays the coordinates for the current window in a multiline edit text box:

```
String strDisplay=""
int rc
long left = 0, top = 0, right = 0, bottom = 0

rc = GetDeskRect(left, top, right, bottom)
strDisplay +=("Desk RECT:~r~n~t Left = " +string(left)&
 +"~r~n~t Top=" + String(top) + "~r~n~t Right = " &
 + String(right)+ "~r~n~t Bottom = " + String(bottom))
mle_1.text = strDisplay
```

See also

GetSIPRect
IsSIPVisible

GetDisplayZoom

Description

Obtains the zoom factor of controls as a percent of their size at design time.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✗

Syntax	**GetDisplayZoom** ( )
Return value	Integer. Returns the current zoom factor for application controls, or -1 if an error occurs or the application is run on the desktop.
Usage	The current zoom factor applies to all controls in an application running on a Windows CE device or emulator. The zoom factor is a percent of the size of the controls at design time. The permissible zoom factor range is 10 to 500 percent.
See also	SetDisplayZoom

GetDynamicDate

Description: Obtains data of type Date from the DynamicDescriptionArea after you have executed a dynamic SQL statement.

PocketBuilder	✗
PowerBuilder	✓

Syntax: *DynamicDescriptionArea*.**GetDynamicDate** (*index*)

Return value: Date. Returns the Date data in the output parameter descriptor identified by *index* in *DynamicDescriptionArea*. Returns `1900-01-01` if an error occurs. If any argument's value is null, GetDynamicDate returns null.

GetDynamicDateTime

Description: Obtains data of type DateTime from the DynamicDescriptionArea after you have executed a dynamic SQL statement.

PocketBuilder	✗
PowerBuilder	✓

Syntax: *DynamicDescriptionArea*.**GetDynamicDateTime** (*index*)

Return value: DateTime. Returns the DateTime data in the output parameter descriptor identified by *index* in *DynamicDescriptionArea*. Returns `1900-01-01 00:00:00.000000` if an error occurs. If any argument's value is null, GetDynamicDateTime returns null.

GetDynamicNumber

Description

Obtains numeric data from the DynamicDescriptionArea after you have executed a dynamic SQL statement.

PocketBuilder	✗
PowerBuilder	✓

Syntax

DynamicDescriptionArea.**GetDynamicNumber** (*index*)

Return value

Double. Returns the numeric data in the output parameter descriptor identified by *index* in *DynamicDescriptionArea*. Returns 0 if an error occurs. If any argument's value is null, GetDynamicNumber returns null.

GetDynamicString

Description

Obtains data of type String from the DynamicDescriptionArea after you have executed a dynamic SQL statement.

PocketBuilder	✗
PowerBuilder	✓

Syntax

DynamicDescriptionArea.**GetDynamicString** (*index*)

Return value

String. Returns the string data in the output parameter descriptor identified by *index* in *DynamicDescriptionArea*. Returns the empty string (`""`) if an error occurs. If any argument's value is null, GetDynamicString returns null.

GetDynamicTime

Description

Obtains data of type Time from the DynamicDescriptionArea after you have executed a dynamic SQL statement.

PocketBuilder	✗
PowerBuilder	✓

Syntax

DynamicDescriptionArea.**GetDynamicTime** (*index*)

Return value

Time. Returns the Time data in the output parameter descriptor identified by *index* in *DynamicDescriptionArea*. Returns `00:00:00.000000` if an error occurs. If any argument's value is null, GetDynamicTime returns null.

GetEnabledDecoders

Description

Retrieves the list of enabled decoders.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✗

Applies to

BarcodeScanner objects

Syntax

Integer *scanner.*GetEnabledDecoders (*intDecoders* [ ])

Argument	Description
scanner	The scanner object that is associated with the scanner for which you want to obtain a list of enabled decoders
intDecoders [ ]	Array of integers that correspond to the decoder IDs of the enabled decoders

Return value

Integer. Returns 1 for success or one of the following negative values if an error occurs:

- **-1** Unspecified error
- **-2** Supporting DLL not loaded error
- **-3** Initialization error other than DLL not loaded
- **-4** Error in the passed in arguments
- **-5** Something in the object instance is inconsistent
- **-6** Call to the driver failed
- **-7** Error opening the specific scan device
- **-8** Error in the internal buffer allocation
- **-9** Incorrect scan state for the requested action (typically benign)
- **-10** Low level device error
- **-100** Feature not implemented

Usage

The enabled decoders are a subset of supported decoders. Decoders must be enabled for use in a scanning operation.

Examples

The following example places the IDs of all the enabled decoders in an array:

```
integer li_rtn, li_firstID, li_secondID, l_IDs[ ]
li_rtn = l_scanner.GetEnabledDecoders(l_IDs)
```

```
   li_firstID = l_IDs[1]
   li_secondID = l_IDs[2]
```

See also

EnableDecoder
GetSupportedDecoders

GetEntry

Retrieves an entry in a call log or a dialing directory.

To	Use
Retrieve an entry in a call log	Syntax 1
Retrieve an entry in a dialing directory	Syntax 2

Syntax 1 For a CallLog object

Description

Retrieves a call log entry based on an index value.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✗

Applies to

CallLog objects

Syntax

objectname.GetEntry (*index*)

Argument	Description
objectname	The name of the call log from which you want to retrieve an entry.
index	An integer that specifies the entry you want to retrieve. Index values are typically in reverse chronological order, such that a value of “1” corresponds to the latest entry.

Return value

A CallLogEntry structure. GetEntry returns a null object when the argument does not correspond to an actual index value.

Examples

These statements get the latest entry in the *l_myCallLog* call log:

```
   Integer l_idx = 1
   CallLogEntry l_mylogentry
   l_mylogentry = l_mycalllog.getEntry (l_idx)
```

See also

GetEntries

Syntax 2 For a DialingDirectory object

Description

Retrieves a dialing directory entry based on an index value and location.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✗

Applies to

DialingDirectory objects

Syntax

objectname.GetEntry (*index*)

Argument	Description
objectname	The name of the dialing directory from which you want to retrieve an entry.
index	An integer that specifies the entry you want to retrieve.

Return value

A DialingDirectoryEntry structure. GetEntry returns a null object when the argument does not correspond to an actual index value.

Examples

The following statements call the getEntry function on the *l_myphonebook* DialingDirectory object to return a DialingDirectoryEntry object:

```
Integer l_idx = 1
DialingDirectoryEntry l_mydirectoryentry
l_mydirectoryentry = l_myphonebook.getEntry (l_idx)
```

See also

GetEntries
UpdateEntry

GetEntries

Retrieves an entire call log or dialing entry into an array.

To	Use
Retrieve a call log into an array of CallLogEntry objects	Syntax 1
Retrieve a dialing directory into an array of DialingDirectoryEntry objects	Syntax 2

Syntax 1 For CallLog objects

Description

Retrieves a call log into an array of CallLogEntry objects.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✗

Applies to

CallLog objects

Syntax

objectname.GetEntries (*entries[ ]*)

Argument	Description
objectname	The name of the call log from which you want to retrieve entries
entries[ ]	An array of CallLogEntry objects returned by reference

Return value

Integer. Returns 1 for success, and one of the following negative values if an error occurs:

-1	Unspecified error
-2	Not Implemented
-3	Supporting DLL not loaded
-4	Error in the arguments passed in
-5	Other initialization error

Examples

The following statements call the getEntries function on the *l_mycalllog* CallLog object to return an array of CallLogEntry objects:

```
// Instance variable:
// CallLogEntry iCallLogEntries[]
CallLog l_mycalllog
l_mycalllog = CREATE CallLog
l_mycalllog.getEntries(iCallLogEntries)
```

See also

GetEntry
UpdateEntry

Syntax 2 For DialingDirectory objects

Description

Retrieves a dialing directory into an array of DialingDirectoryEntry objects.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✗

Applies to

DialingDirectory objects

Syntax

objectname.GetEntries (*entries[ ]*)

Argument	**Description**
objectname	The name of the dialing directory from which you want to retrieve entries
entries[ ]	An array of DialingDirectoryEntry objects returned by reference

Return value

Integer. Returns 1 for success, and one of the following negative values if an error occurs:

-1 Unspecified error

-2 Not Implemented

-3 Supporting DLL not loaded

-4 Error in the arguments passed in

-5 Other initialization error

Examples

The following statements call the getEntries function on the *l_myphonebook* DialingDirectory object to return an array of DialingDirectoryEntry objects:

```
// Instance variable:
// DialingDirectoryEntry iDialingDirEntries[]
DialingDirectory l_myphonebook
l_myphonebook = CREATE DialingDirectory
l_myphonebook.getEntries(IDialingDirEntries)

l_mydirectoryentry = l_myphonebook.getEntry (l_idx)
```

See also

GetEntry

GetEnvironment

Description

Gets information about the operating system, processor, and screen display of the system.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

GetEnvironment (*environmentinfo*)

Argument	Description
environmentinfo	The name of the Environment object that will hold the information about the environment

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs. If *environmentinfo* is null, GetEnvironment returns null.

Usage

In cross-platform development projects, you can call GetEnvironment in scripts and take actions based on the operating system. You can also find out the processor (Intel 386 or 486, 68000, and so on). The information also includes version numbers of the operating system and PocketBuilder.

You can call GetEnvironment to find out the number of colors supported by the system and the size of the screen. You can use the size information in a window's Open script to reset its X and Y properties.

Examples

This script runs another PocketBuilder application and uses the OSType property of the Environment object to determine how to specify the path:

```
string path
environment env
integer rtn

rtn = GetEnvironment(env)
IF rtn <> 1 THEN RETURN

CHOOSE CASE env.OSType
CASE Windows!, WindowsNT!
    path = "C:\PB_apps\analyze.exe"
CASE WindowsCE!
    path = "\windows\analyze.exe"
CASE ELSE
    RETURN
END CHOOSE
Run(path)
```

This example displays a message box that shows the major, minor, and fixes versions of PocketBuilder:

```
string ls_version
environment env
integer rtn

rtn = GetEnvironment(env)

IF rtn <> 1 THEN RETURN
ls_version = "Version: "+ string(env.pbmajorrevision)
ls_version += "." + string(env.pbminorrevision)
ls_version += "." + string(env.pbfixesrevision)
ls_version += " Build: " + string(env.pbbuildnumber)

MessageBox("PocketBuilder Version", ls_version)
```

GetFileOpenName

Description

Displays the system's Open File dialog box and allows the user to select a file or enter a file name.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

GetFileOpenName (*title*, *pathname*, *filename* {, *extension* {, *filter* { , *initdir* { , *aFlag* } } } })

GetFileOpenName (*title*, *pathname*, *filename[ ]* {, *extension* {, *filter* { , *initdir* { , *aFlag* } } } })

Argument	Description
title	A string whose value is the title of the dialog box.
pathname	A string variable in which you want to store the returned path and file name. On a handheld device, the *pathname* value must be the *My Documents* directory or one of its subdirectories.
filename, *filename[ ]*	A string variable in which the returned file name is stored or an array of string variables in which multiple selected file names are stored. Specifying an array of string variables enables multiple selection in the dialog box.

Argument	Description
extension (optional)	A string whose value is a 1- to 3-character default file extension. The default is no extension.
filter (optional)	A string whose value is a text description of the files to include in the list box and the file mask that you want to use to select the displayed files (for example, *.* or *.exe). The format for *filter* is: *description*,*. *ext* The default is: `"All Files (*.*),*.*"`
initdir (optional)	A string whose value is the initial directory name. The default is the current directory.
aFlag (optional)	An unsigned long whose value determines which options are enabled in the dialog box. The value of each option's flag is calculated as 2 to the power of (*index* -1), where *index* is the integer associated with the option. The value of the aggregate flag passed to GetFileOpenName is the sum of the individual option flags. See the table in the Usage section for a list of options, the index associated with each option, and the option's meaning.

Return value

Integer. Returns 1 if it succeeds, 0 if the user clicks the Cancel button or Windows cancels the display, and -1 if an error occurs. If any argument's value is null, GetFileOpenName returns null.

Usage

If you specify a DOS-style file extension and the user enters a file name with no extension, PocketBuilder appends the default extension to the file name. If you specify a file mask to act as a filter, PocketBuilder displays only files that match the mask.

You use the *filter* argument to limit the types of files displayed in the list box and to let the user know what those limits are. For example, to display the description Text Files (*.*TXT*) and only files with the extension .*TXT*, specify the following for *filter*:

```
"Text Files (*.TXT),*.TXT"
```

To specify more than one file extension in *filter*, enter multiple descriptions and extension combinations and separate them with commas. For example:

```
"PIF files, *.PIF, Batch files, *.BAT"
```

The dialog boxes presented by GetFileOpenName and GetFileSaveName are system dialog boxes. They provide standard system behavior, including control over the current directory. When users change the drive, directory, or folder in the dialog box, they change the current directory or folder. The newly selected directory or folder becomes the default for file operations until they exit the application, unless the optional *initdir* argument is passed.

The *aFlag* argument is used to pass one or more options that determine the appearance of the dialog box. For each option, the value of the flag is `2^(`***`index`***` -1)`, where *index* is an integer associated with each option as shown in the following table. You can pass multiple options by passing an aggregate flag, calculated by adding the values of the individual flags.

If you do not pass an *aFlag*, the Explorer-style open file dialog box is used. If you do pass a flag, the old-style dialog box is used by default. Some options do not apply when the Explorer-style dialog box is used. For those that do apply, add the option value for using the Explorer-style dialog box (2) to the value of the option if you want to display an Explorer-style dialog box.

For example, passing the flag 32768 (2^15) to the GetFileSaveName function opens the old-style dialog box with the Read Only check box selected by default. Passing the flag 32770 opens the Explorer-style dialog box with the Read Only check box selected by default.

Table 10-4: Option values for GetFileOpenName and GetFileSaveName

Index	Constant name	Description
1	OFN_CREATEPROMPT	If the specified file does not exist, prompt for permission to create the file. If the user chooses to create the file, the dialog box closes; otherwise the dialog box remains open.
2	OFN_EXPLORER	Use an Explorer-style dialog box.
3	OFN_EXTENSIONDIFFERENT	The file extension entered differed from the extensions specified in extension.
4	OFN_FILEMUSTEXIST	Only the names of existing files can be entered.
5	OFN_HIDEREADONLY	Hide the Read Only check box.
6	OFN_LONGNAMES	Use long file names. Ignored for Explorer-style dialog boxes.
7	OFN_NOCHANGEDIR	Restore the current directory to its original value if the user changed the directory while searching for files. This option has no effect for GetFileOpenName on desktop operating systems.
8	OFN_NODEREFERENCELINKS	Return the path and file name of the selected shortcut (*.lnk* file); otherwise the path and file name pointed to by the shortcut are returned.
9	OFN_NOLONGNAMES	Use short file names (8.3 format). Ignored for Explorer-style dialog boxes.
10	OFN_NONETWORKBUTTON	Hide the Network button. Ignored for Explorer-style dialog boxes.
11	OFN_NOREADONLYRETURN	The file returned is not read only and is not in a write-protected directory.

Index	Constant name	Description
12	OFN_NOTESTFILECREATE	Do not create the file before the dialog box is closed. This option should be specified if the application saves the file on a netwrok share where files can be created but not modified. No check is made for write protection, a full disk, an open drive door, or network protection. A file cannot be reopened once it is closed.
13	OFN_NOVALIDATE	Invalid characters are allowed in file names.
14	OFN_OVERWRITEPROMPT	Used in Save As dialog boxes. Generates a message box if the selected file already exists.
15	OFN_PATHMUSTEXIST	Only valid paths and file names can be entered.
16	OFN_READONLY	Select the Read Only check box when the Save dialog box is created.

Opening a file
Use the FileOpen function to open a selected file.

Examples

In the following example, the dialog box has the title Open and displays text files, batch files, and INI files in the Files of Type drop-down list.

```
// instance variables
// string is_filename, is_fullname
int   li_fileid

if GetFileOpenName ("Open", is_fullname, is_filename, &
   "txt", "Text Files (*.txt),*.txt,INI Files " &
   + "(*.ini), *.ini,Batch Files (*.bat),*.bat") &
   < 1 then return

li_fileid = FileOpen (is_fullname, StreamMode!)
FileRead (li_fileid, mle_notepad.text)
FileClose (li_fileid)
```

See also

DirList
DirSelect
GetFileSaveName

GetFileSaveName

Description

Displays the system's Save File dialog box with the specified file name displayed in the File name box. The user can enter a file name or select a file from the grayed list.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

GetFileSaveName (*title*, *pathname*, *filename* {, *extension* {, *filter* { , *initdir* { , *aFlag* } } } })

GetFileSaveName (*title*, *pathname*, *filename [ ]* {, *extension* {, *filter* { , *initdir* { , *aFlag* } } } })

Argument	Description
title	A string whose value is the title of the dialog box.
pathname	A string variable whose value is the default file name and which stores the returned path and file name. On a handheld device, the *pathname* value must be the *My Documents* directory or one of its subdirectories. The default file name is displayed in the File name box; the user can specify another name.
filename, *filename[ ]*	A string variable in which the returned file name is stored or an array of string variables in which multiple selected file names are stored. Specifying an array of string variables enables multiple selection in the dialog box.
extension (optional)	A string whose value is a 1- to 3-character default file extension. The default is no extension.
filter (optional)	A string whose value is the description of the displayed files and the file extension that you want use to select the displayed files (the filter). The format for *filter* is: *description*,*. *ext* The default is: `"All Files (*.*),*.*"`
initdir (optional)	A string whose value is the initial directory name. The default is the current directory.
aFlag (optional)	An unsigned long whose value determines which options are enabled in the dialog box. The value of each option's flag is calculated as 2 to the power of (*index* -1), where *index* is the integer associated with the option. The value of the aggregate flag passed to GetOpenFileName is the sum of the individual option flags. See the table in the Usage section for GetOpenFileName for a list of options, the index associated with each option, and the option's meaning.

Return value

Integer. Returns 1 if it succeeds, 0 if the user clicks the Cancel button or Windows cancels the display, and -1 if an error occurs. If any argument's value is null, GetFileSaveName returns null.

Usage

If you specify a DOS-style extension and the user enters a file name with no extension, PocketBuilder appends the default extension to the file name. If you specify a file mask to act as a filter, PocketBuilder displays only files that match the mask.

For usage notes on the *filter*, *initdir*, and *aFlag* arguments, see the GetFileOpenName function.

Examples

These statements display the Select File dialog box. The default file extension is *.DOC* and the filter is all files. If a file is selected successfully, its path displays in a SingleLineEdit control:

```
string ls_path, ls_file
int li_rc

ls_path = sle_1.Text
li_rc = GetFileSaveName ( "Select File", &
   ls_path, ls_file, "DOC", &
   "All Files (*.*),*.*")

IF li_rc = 1 Then
   sle_1.Text = ls_path + "\" + ls_file
End If
```

See also

GetFileOpenName
DirList
DirSelect

GetFirstSheet

Description

Obtains the top sheet in the MDI frame, which may or may not be active.

PocketBuilder	✗
PowerBuilder	✓

Applies to

MDI frame windows

Syntax

mdiframewindow.**GetFirstSheet** ( )

Return value

Window. Returns the first (top) sheet in the MDI frame. If no sheet is open in the frame, GetFirstSheet returns an invalid value. If *mdiframewindow* is null, GetFirstSheet returns null.

GetFix

Description

Populates a GPSFix structure with data from the current position fix.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✗

Applies to

GPS and SerialGPS objects

Syntax

Integer *GPSname*.GetFix (*myGPSFix*)

Argument	Description
GPSname	Name of the GPS or SerialGPS object
myGPSFix	GPSFix structure object passed by reference that can store the data from the current position fix

Return value

Integer. Returns 1 for success and 100 or a negative number for an error. The following is a list of possible error codes and their meanings:

100 End of buffer. The requested data was not found.

-1 General error.

-10 Invalid object. Could occur if the GPS object instance is corrupted.

-13 Not previously opened. This function cannot be called until a GPS object or SerialGPS object has been successfully opened.

-14 Read timeout. Occurs when the timeout interval (a ConfigParams property of the SerialGPS object) is exceeded.

-15 Read Failure. Unable to read the file or serial port.

-16 Parser Error. Parser is unable to interpret a sentence. This error is generated when nonstandard tokens are discovered while parsing the GPS data.

-17 Checksum Error. Most GPS sentences end in a two-digit checksum value. The PocketBuilder parser verifies this value and reports a checksum error if the calculated value does not match the stated value.

Usage

Use this function to populate a GPSFix structure with data about the current position fix, including data about the location of the fix and how reliable the data is. Each GetFix request obtains data from a different $GPGGA sentence in the data buffer. If the end of the data buffer is reached before finding a new sentence to parse, a GPS object returns 100 to indicate the end of the buffer has been reached. A SerialGPS object automatically reads in a new buffer and searches the new buffer. If this second data buffer does not contain a $GPGGA sentence, then the SerialGPS object returns 100.

Examples

The following lines create a SerialGPS object, retrieve information about the current position fix, test the validity of the GPSFix object, and write data to a multiline edit box:

```
SerialGps myGPS
GPSFix myFix
Real lr_alt, lr_gh, lr_hdop
Integer li_numsats, rc

MyGPS = CREATE SerialGPS
myGPS.Open()
...
rc = MyGPS.GetFix(myFix)
if rc = 1 then
  if myFix.IsFixValid then
    lr_alt = myFix.Altitude
    lr_gh = myFix.geoidalheight
    lr_hdop = myFix.HDOP
    mle_fix.text = "Recorded at: " +  &
       String(myFix.FixTime)
    mle_fix.text += "Altitude: " + String(lr_Alt)
    mle_fix.text += "Geoidal height :" + String(lr_gh)
    mle_fix.text += "HDOP: " + String(lr_hdop)
    mle_fix.text += "Satellites: " + String(li_numsats)
  else
    return -1
  end if
else
  return -1
end if
```

The Latitude and Longitude properties of the GPSFix structure take a value of the GPSCoordinate structure. The following example shows how you might extend the previous example to display the Longitude property value in the multiline edit box. It takes the Minute property, separates it into whole minutes and a partial minute, and converts the partial minute into a number of seconds:

```
GPSCoordinate myLongCoord
```

```
Integer fixLongMins, rc
Real fixLongSecs
...
rc = MyGPS.GetFix(myFix)
myLongCoord = myFix.Longitude
fixLongMins = Integer(myLongCoord.Minute)
fixLongSecs = (myLongCoord.Minute - fixLongMins) * 60
mle_fix.text = "Longitude: " &
   + String(myLongCoord.degree) + " degrees "  &
   + String(fixLongMins) + " minutes "  &
   + String(fixLongSecs) + " seconds "  &
   + String(myLongCoord.Hemisphere)
```

See also

GetHeading
GetSatellitesInView
Open

GetFixesVersion

Description

Returns the fix level for the current PocketBuilder execution context. For example, at maintenance level 1.5.2, the fix version is 2.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

ContextInformation objects

Syntax

servicereference.**GetFixesVersion** (*fixversion*)

Argument	Description
servicereference	Reference to the ContextInformation service instance.
fixversion	Integer into which the function places the fix version. This argument is passed by reference.

Return value

Integer. Returns 1 if the function succeeds and -1 if an error occurs.

Usage

Call this function to determine the current fix version.

Examples

This example calls the GetFixesVersion function:

```
String ls_name
Constant String ls_currver = "8.0.3"
```

```
Integer li_majver, li_minver, li_fixver
ContextInformation ci

this.GetContextService ("ContextInformation", ci)
ci.GetMajorVersion(li_majver)
ci.GetMinorVersion(li_minver)
ci.GetFixesVersion(li_fixver)
IF li_majver <> 8 THEN
    MessageBox("Error", &
      "Must be at Version " + ls_currver)
ELSEIF li_minver <> 0 THEN
    MessageBox("Error", &
      "Must be at Version " + ls_currver)
ELSEIF li_fixver <> 3 THEN
    MessageBox("Error", &
      "Must be at Version " + ls_currver)
END IF
```

See also

GetCompanyName
GetHostObject
GetMajorVersion
GetMinorVersion
GetName
GetShortName
GetVersionName

GetFocus

Description

Determines the control that currently has focus.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

GetFocus ( )

Return value

GraphicObject. Returns the control that currently has focus. Returns an invalid control reference if an error occurs.

Use the IsValid function to determine whether GetFocus has returned a valid control.

Examples

These statements set *which_control* equal to the datatype of the control that currently has focus, and then set *text_value* to the text property of the control:

```
GraphicObject which_control
SingleLineEdit sle_which
CommandButton cb_which
string text_value

which_control = GetFocus()

CHOOSE CASE TypeOf(which_control)

CASE CommandButton!
    cb_which = which_control
    text_value = cb_which.Text

CASE SingleLineEdit!
    sle_which = which_control
    text_value = sle_which.Text

CASE ELSE
    text_value = ""
END CHOOSE
```

See also

IsValid
SetFocus

GetFolder

Description

Displays a folder selection dialog box.

PocketBuilder	✗
PowerBuilder	✓

Syntax

GetFolder (*title*, *directory*)

Return value

Integer. Returns 1 if the function succeeds, 0 if the user selects cancel (or the dialog box is closed), -1 if an error occurs.

GetGlobalProperty

Description
Returns the value of an SSL global property. This function is used by PowerBuilder clients connecting to EAServer.

PocketBuilder	✗
PowerBuilder	✓

Applies to
SSLServiceProvider object

Syntax
sslserviceprovider.**GetGlobalProperty** (*property, values*)

Return value
Long. Returns 0 for success and a negative value if an error has occurs.

GetHeading

Description
Populates a GPSHeading structure with data from the current heading.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✗

Applies to
GPS and SerialGPS objects

Syntax
Integer *GPSname*.GetHeading(*GPSHeading*)

Argument	Description
GPSname	Name of the GPS or SerialGPS object
GPSHeading	Structure passed by reference that stores speed and directional information used by the SerialGPS object

Return value
Integer. Returns 1 for success and 100 or a negative number for an error. The following is a list of possible error codes and their meanings:

100 End of buffer. The requested data was not found.

-1 General error.

-10 Invalid object. Could occur if the GPS object instance is corrupted.

-13 Not previously opened. This function cannot be called until a GPS object or SerialGPS object has been successfully opened.

-14 Read timeout. Occurs when the timeout interval (a ConfigParams property of the SerialGPS object) is exceeded.

-15 Read Failure. Unable to read the file or serial port.

-16 Parser Error. Parser is unable to interpret a sentence. This error is generated when nonstandard tokens are discovered while parsing the GPS data.

-17 Checksum Error. Most GPS sentences end in a two-digit checksum value. The PocketBuilder parser verifies this value and reports a checksum error if the calculated value does not match the stated value.

Usage

Use this function to populate a GPSHeading structure with information about the direction of travel, ground speed, and magnetic variation.

Examples

The following lines create a SerialGPS object, retrieve information about the current position fix, test the validity of the GPSHeading object, and write data to a multiline edit box:

```
SerialGps myGPS
real TrueHeading, Speed, MV
char MVD
GPSHeading myGPSHeading
Integer rc

MyGPS = CREATE SerialGPS
myGPS.Open()
...
rc = MyGPS.GetHeading(myGPSHeading)
IF rc = 1 THEN
TrueHeading = myGPSHeading.Heading
Speed = myGPSHeading.groundspeed
MV = myGPSHeading.MagneticVariation
MVD = myGPSHeading.MagneticVariationDirection
mle_1 = "Ground speed: " + String(Speed)
mle_1 += "True heading: " + String(TrueHeading) + "~r~n"
mle_1 += "Variation: " + String(MV) + MVD
ELSE
//Process error
END IF
```

See also

GetFix
GetSatellitesInView
Open

GetHostObject

Description

Provides a reference to the context's host object.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

ContextInformation objects

Syntax

servicereference.**GetHostObject** (*hostobject*)

Argument	Description
servicereference	Reference to the Context Information service instance
hostobject	PowerObject into which the function places a reference to the ActiveX automation server object

Return value

Integer. Returns 1 if the function succeeds and -1 if an error occurs. In PocketBuilder applications, GetHostObject always returns -1.

Usage

Call this function to obtain a reference to the context object model.

PocketBuilder environments
The host object in a PocketBuilder application is an empty object—it fails the isValid(obj) test.

See also

GetCompanyName
GetName
GetShortName
GetVersionName

GetItem

Retrieves data associated with a specified item in ListView, TreeView, and Toolbar controls.

To retrieve data associated with a specified	Use
ListView control item	Syntax 1
ListView control item and column	Syntax 2
TreeView item	Syntax 3
Toolbar control item	Syntax 4

Syntax 1 For ListView controls

Description

Retrieves a ListViewItem object from a ListView control so you can examine its properties.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

ListView controls

Syntax

listviewname.**GetItem** (*index*, {*column*}, *item*)

Argument	Description
listviewname	The name of the ListView control for which you want to retrieve the ListView item
index	The index number of the item you want to retrieve
column	The index number of the column for which you want item information
item	The ListViewItem variable in which you want to store the ListViewItem object

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs. Stores a ListViewItem object in a ListViewItem variable.

Usage

You can retrieve properties for any ListView item with this syntax. If you do not specify a column, GetItem retrieves properties for the first column of an item. Only report views display multiple columns.

To retrieve labels only, use syntax 2. You can use GetColumn to obtain column properties that are not specific to a ListView item.

To change pictures and other property values associated with a ListView item, use GetItem, change the property values, and use SetItem to apply the changes back to the ListView.

Examples

This example uses GetItem to move the second item in the lv_list ListView control to the fifth item. It retrieves item 2, inserts it into the ListView control as item 5, and then deletes the original item:

```
listviewitem l_lvi

lv_list.GetItem(2, l_lvi)
lv_list.InsertItem(5, l_lvi)
lv_list.DeleteItem(2)
```

See also

GetColumn
SetItem

Syntax 2 For ListView controls

Description

Retrieves the value displayed for a ListView item in a specified column.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

ListView controls

Syntax

listviewname.**GetItem** (*index, column, label*)

Argument	Description
listviewname	The name of the ListView control from which you want to retrieve a displayed value.
index	The index number of the item for which you want to retrieve a displayed value.
column	The index number of the column for which you want to retrieve a value. If the ListView is not a multicolumn report view, all the items are considered to be in column 1.
label	A string variable in which you store the displayed value.

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs. Stores the displayed value of the ListView column in a string variable.

Usage

To retrieve property values for a ListView item, use Syntax 1.

Examples

This example gets the displayed values from column 1 and column 3 of the first row of the lv_list ListView and displays them in the sle_info SingleLineEdit control.

```
string ls_artist, ls_comp

lv_list.GetItem(1, 1 , ls_comp)
lv_list.GetItem(1, 3 , ls_artist)
sle_info.text = ls_artist +" wrote " + ls_comp + "."
```

See also

SetItem

Syntax 3 For TreeView controls

Description

Retrieves the data associated with the specified item.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

TreeView controls

Syntax

treeviewname.**GetItem** (*itemhandle, item*)

Argument	Description
treeviewname	The name of the TreeView control in which you want to get data for a specified item
itemhandle	The handle for the item for which you want to retrieve information
item	A TreeViewItem variable in which you want to store the item identified by the item handle

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs.

Usage

Use GetItem to retrieve the state information associated with a specific item in a TreeView (such as label, handle, or picture index). After you have retrieved the information, you can use it in your application. To change a property of an item, call GetItem to assign the item to a TreeViewItem variable, change its properties, and call SetItem to copy the changes back to the TreeView.

Examples

This code for the Clicked event gets the clicked item and changes it overlay picture. The SetItem function copies the change back to the TreeView:

```
treeviewitem tvi
This.SetItem(handle, tvi)
tvi.OverlayPictureIndex = 1
This.SetItem(handle, tvi)
```

This example tracks items in the SelectionChanged event. If there is no prior selection, the value of *l_tviold* is zero:

```
treeviewitem l_tvinew, l_tviold

// Get the treeview item that was the old selection
tv_list.GetItem(oldhandle, l_tviold)

// Get the treeview item that is currently selected
tv_list.GetItem(newhandle, l_tvinew)

// Print the labels for the two items in the
// SingleLineEdit
sle_get.Text = "Selection changed from " &
    + String(l_tviold.Label) + " to " &
    + String(l_tvinew.Label)
```

See also

InsertItem

Syntax 4 For Toolbar controls

Description

Gets a reference to an item in the toolbar.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✗

Applies to

Toolbar controls

Syntax

Integer *controlname*.GetItem (*toolbarindex*, *item*)

Argument	Description
controlname	The name of the toolbar control
toolbarindex	Integer for the index of the toolbar item
item	Reference to a ToolbarItem object

Return value

Integer. Returns 1 for success and -1 if an error occurs.

Examples

The following example passes a reference to the second item in the toolbar:

```
Integer li_rtn
ToolbarItem myItem
myItem = CREATE ToolbarItem
li_rtn = tlbr_mytoolbar.GetItem(2, myItem)
```

See also

AddItem
DeleteItem
InsertItem

GetItemAtPointer

Description

Gets the handle or the index of the item under the cursor.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

ListView controls, TreeView controls

Syntax

controlname.**GetItemAtPointer** ( )

Argument	Description
controlname	The name of the control whose handle or index you want to obtain.

Return value

Long. Returns the index (ListView) or handle (TreeView) of the item under the cursor. Returns -1 for failure.

Usage

System events that select an item in a ListView or TreeView control, such as the Clicked event, already have an argument that passes the index for the ListView or the handle for the TreeView. The GetItemAtPointer function allows you to retrieve the index or handle in user events (or system events without an index or handle argument) for a ListView or TreeView control.

Examples

This example places the handle of a TreeView item in a SingleLineEdit box:

```
integer li_index

li_index= tv_1.GetItematPointer ( )
sle_1.text = string (li_index)
```

See also

FindItem
SelectItem

GetItemPictureIndex

Description Gets the picture index that corresponds to the item index of a toolbar item.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✗

Applies to Toolbar controls

Syntax Integer *controlname*.GetItemPictureIndex (*toolbarindex*)

Argument	Description
controlname	The name of the toolbar control
toolbarindex	Integer for the index of the toolbar item

Return value Integer. Returns the picture index that identifies the picture associated with the toolbar item defined by the value of the *toolbarindex* argument. Returns -1 if an error occurs.

Examples The following example gets the picture index for the second item in the toolbar and places it in a local variable:

```
Integer li_picindex
li_picindex = tlbr_mytoolbar.GetItemPictureIndex(2)
```

See also SetItemPictureIndex

GetItemState

Description Gets the state of a toolbar item.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✗

Applies to Toolbar controls

Syntax

Integer *controlname*.GetItemState (*toolbarindex*)

Argument	Description
controlname	The name of the toolbar control
toolbarindex	Integer for the index of the toolbar item

Return value

Integer. Values can be any of the values described in the following table, or combinations of these values:

Value	Windows CE value	Description
1	TBSTATE_CHECKED	The toolbar button has a StyleCheck! or StyleCheckGroup! style and remains in the depressed state
2	TBSTATE_PRESSED	The toolbar button has a StyleButton! style and is temporarily in the depressed state
4	TBSTATE_ENABLED	The toolbar button is enabled for selection
32	TBSTATE_WRAP	The next item in the toolbar that is not grouped with the current toolbar button is on a separate line

Values are additive. For example, a toolbar button with the checked state (1) can also be enabled (4) and wrapped (32). In this case the return value would be 37.

Returns -1 if an error occurs.

Examples

The following example gets the state for the second item in the toolbar and places it in a local variable:

```
integer li_picstate
li_picstate = tlbr_mytoolbar.GetItemState(2)
```

See also

SetItemState

GetLastReturn

Description	Returns the return value from the last InvokePBFunction or TriggerPBEvent function.

PocketBuilder	✗
PowerBuilder	✓

Applies to	Window ActiveX controls
Syntax	*activexcontrol*.**GetLastReturn** ( )
Return value	Any. Returns the last return value.

GetLibraryList

Description	Gets the files in the library search path of the application.

PocketBuilder	✗
PowerBuilder	✓

Syntax	**GetLibraryList** ( )
Return value	String. Returns the current library list with complete paths. Multiple libraries are separated by commas.

GetMajorVersion

Description

Returns the major version for the current PocketBuilder execution context. For example, at maintenance level 1.5.2 the major version is 1.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

ContextInformation objects

Syntax

servicereference.**GetMajorVersion** (*majorversion*)

Argument	Description
servicereference	Reference to the ContextInformation service instance.
majorversion	Integer into which the function places the major version. This argument is passed by reference.

Return value

Integer. Returns 1 if the function succeeds and -1 if an error occurs.

Usage

Call this function to determine the current major version.

Examples

This example calls the GetMajorVersion function:

```
String ls_name
Constant String ls_currver = "8.0.3"
Integer li_majver, li_minver, li_fixver
ContextInformation ci

this.GetContextService ("ContextInformation", ci)

GetMajorVersion(li_majver)
ci.GetMinorVersion(li_minver)
ci.GetFixesVersion(li_fixver)
IF li_majver <> 8 THEN
   MessageBox("Error", &
      "Must be at Version " + ls_currver)
ELSEIF li_minver <> 0 THEN
   MessageBox("Error", &
      "Must be at Version " + ls_currver)
ELSEIF li_fixver <> 3 THEN
   MessageBox("Error", &
      "Must be at Version " + ls_currver)
END IF
```

See also

GetCompanyName
GetFixesVersion

GetHostObject
GetMinorVersion
GetName
GetShortName
GetVersionName

GetMessage

Description

Returns the error message from objects of type Throwable.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

throwableobject.**GetMessage** ( )

Argument	Description
throwableobject	Object of type Throwable from which you want to retrieve an error message

Return value

String. The error text for system error objects, such as RuntimeError, is preset.

Usage

You can set the error message for an object of type Throwable using the SetMessage function.

Examples

This example catches a system error message and displays that error in a message box. Catching the system error prevents the application from terminating when the arccosine argument, entered by the application user, is not in the required range:

```
Double ld_num
ld_num = Double (sle_1.text)

TRY
sle_2.text = string (acos (ld_num))
CATCH (runtimeerror er)
   MessageBox("Runtime Error", er.GetMessage())
END TRY
```

This example catches and displays a user error message from the Clicked event of a button that calls the user-defined function, wf_acos. The user-defined function catches a runtime error—preventing the application from terminating—and then sets the message for a user object, uo_exception, that inherits from the Exception object type:

```
TRY
   wf_acos()
CATCH (uo_exception u_ex)
   messageBox("Out of Range", u_ex.GetMessage())
END TRY
```

Code for the wf_acos function is shown in the SetMessage function.

See also

SetMessage

GetMessageStatus

Description

Retrieves information about an SMS message sent during the current session.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✗

Applies to

SMSSession objects

Syntax

objectname.GetMessageStatus (*smsmsg*)

Argument	Description
objectname	The name of the SMSSession object
smsmsg	An SMS message structure returned by reference that contains information about the message

Return value

Integer. Returns 1 for success and a negative value if an error occurs.

Usage

The GetMessageStatus function retrieves an SMSMessage structure that contains information about the message's ID, options, validity period, text content if any, and status.

Examples

The following example retrieves information about a message in the global variable *g_smsMsg*, stores the value of its Status property in the *msgStat* SMSMsgStatus variable, and writes the value of *msgStat* to a single line edit box:

```
// Global variables:
// SMSSession g_smsSess
// SMSMessage g_smsMsg

SMSMsgStatus msgStat
integer li_ret

li_ret = g_smsSess.GetMessageStatus(g_smsMsg)
msgStat = g_smsMsg.Status
sle_status.text = "Message status: " + String(msgStat)
```

See also

Open
Send

GetMinorVersion

Description

Returns the minor version for the current PocketBuilder execution context. For example, at maintenance level 1.5.2 the minor version is 5.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

ContextInformation objects

Syntax

servicereference.**GetMinorVersion** (*minorversion*)

Argument	Description
servicereference	Reference to the ContextInformation service instance.
minorversion	Integer into which the function places the minor version. This argument is passed by reference.

Return value

Integer. Returns 1 if the function succeeds and -1 if an error occurs.

Usage

Call this function to determine the current minor version.

Examples

This example calls the GetMinorVersion function:

```
String ls_name
Constant String ls_currver = "8.0.3"
Integer li_majver, li_minver, li_fixver
ContextInformation ci

this.GetContextService("ContextInformation", ci)

ci.GetMajorVersion(li_majver)
ci.GetMinorVersion(li_minver)
ci.GetFixesVersion(li_fixver)
IF li_majver <> 8 THEN
   MessageBox("Error", &
      "Must be at Version " + ls_currver)
ELSEIF li_minver <> 0 THEN
   MessageBox("Error", &
      "Must be at Version " + ls_currver)
ELSEIF li_fixver <> 3 THEN
   MessageBox("Error", &
      "Must be at Version " + ls_currver)
END IF
```

See also

GetCompanyName
GetFixesVersion
GetHostObject
GetMajorVersion
GetName
GetShortName
GetVersionName

GetName

Description

Gets the name for the current execution context.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

ContextInformation objects

Syntax

servicereference.**GetName** (*name*)

Argument	Description
servicereference	Reference to the ContextInformation service instance.
name	String into which the function places the name. This argument is passed by reference.

Return value

Integer. Returns 1 if the function succeeds and -1 if an error occurs. The function returns values as follows:

- **PocketBuilder runtime:** PocketBuilder Runtime
- **PowerBuilder runtime:** PowerBuilder Runtime
- **PowerBuilder window plug-in:** PowerBuilder window Plug-in
- **PowerBuilder window ActiveX:** PowerBuilder Runtime ActiveX

Usage

Call this function to determine the current execution environment.

Examples

This example calls the GetName function. *ci* is an instance variable of type ContextInformation:

```
String ls_name

this.GetContextService("ContextInformation", ci)
ci.GetName(ls_name)
IF ls_name <> "PocketBuilder Runtime" THEN
   cb_close.visible = FALSE
END IF
```

See also

GetCompanyName
GetContextService
GetFixesVersion
GetHostObject
GetMajorVersion
GetMinorVersion
GetShortName
GetVersionName

GetNativePointer

Description	Gets a pointer to the OLE object associated with the OLE control. The pointer lets you call OLE functions in an external DLL for the object.

PocketBuilder	✗
PowerBuilder	✓

Applies to	OLE controls and OLE custom controls
Syntax	*olename*.**GetNativePointer** (*pointer*)
Return value	Integer. Returns 0 if it succeeds and -1 if an error occurs.

GetNextSheet

Description	Obtains the sheet that is behind the specified sheet in the MDI frame.

PocketBuilder	✗
PowerBuilder	✓

Applies to	MDI frame windows
Syntax	*mdiframewindow*.**GetNextSheet** (*sheet*)
Return value	Window. Returns the sheet that is behind *sheet* in the MDI frame. If there is no sheet behind *sheet*, GetNextSheet returns an invalid value. If any argument's value is null, GetNextSheet returns null.

GetObjectRevisionFromRegistry

Description	Reserved for future use. Assigns synchronization property values saved in the Windows registry to a synchronization object.
Applies to	MLSynchronization, MLSync controls
Syntax	*SyncObject*.**GetObjectRevisionFromRegistry** ( )

Argument	Description
syncObject	The name of the synchronization object

Return value

Integer. Returns the value of ObjectRevision. Returns -1 if the registry key is not found or if the SyncRegistryKey property of the synchronization object is not set.

GetOption

Description

Obtains the value of a specific option for a camera device.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✗

Applies to

Camera objects

Syntax

objectname.GetOption (*Opt*)

Argument	Description
objectname	The name of the camera object that you want to inquire about
Opt	A value of the CameraOption enumerated variable that specifies the name of the option that you want to want to inquire about. For a list of options, see *Usage*

Return value

Integer. Returns the value of the option or 0 if the option is not supported on the device.

Usage

Use the GetOption function to obtain the value of a specific option. Camera options are settings available on various devices. Not all options are available on all devices. Most options are cached and used only when applicable. Some options, such as CamOptPowerUp, are acted on immediately.

The following table lists the available options and their settings. The third column in the table lists some of the devices that support each option. Note that support for devices is limited in this release, and not all listed devices are supported.

Option	Setting	Example of supported device
CamOptAEMetering!	Automatic Exposure Metering points. Values are: 0 = full picture averaging 1 = center weighted 2 = center spot	HP PhotoSmart VEO 130S

Option	Setting	Example of supported device
CamOptBrightness!	Brightness of the image. Values are: 0 = low exposure 1000 = high exposure	Hitachi G1000 LifeView FlyCam CF
CamOptCaptureFormat!	Format of captured image. On some devices, JPEG is the only format supported. Values are: 1 = JPEG (default) 2 = MPEG4	
CamOptCaptureMode!	Capture mode. Values are: 1 = static image (default) 2 = video	
CamOptColorMode!	Color of the picture. Values are: 0 = full color 1 = black and white 2 = negative 4 = “cool” colors	HP PhotoSmart VEO 130S
CamOptContrast!	Image contrast for adjacent areas of the image. The value is an integer in the range 0 to 1000.	LifeView FlyCam CF
CamOptFlash!	Whether the flash should be fired when capturing the image.Values are: 0 = clear flash mode 1 = set flash mode	LifeView FlyCam CF
CamOptFlashDistance!	The distance from the flash to the subject. Values are: Flash_50cm, Flash_100cm, Flash_150cm, or Flash_300cm	LifeView FlyCam CF
CamOptFlickerFrequency!	Sets the flicker filter frequency. Values are: 0 = Automatic (not supported on all devices) 50 = 50Hz 60 = 60Hz (default)	LifeView FlyCam CF
CamOptGamma!	Amount of gamma correction applied to the luminance values of the picture. The value is an integer in the range 0 to 1000.	
CamOptHue!	The quality of a color as determined by its dominant wavelength. The value is an integer in the range -180 to 180 with a default of 0.	LifeView FlyCam CF
CamOptLuminosity!	Adjusts image to compensate for the amount of light emitted by the subject. The value is an integer in the range 0 to 1000.	LifeView FlyCam CF
CamOptMoonLight!	Sets or clears night vision (moonlight) mode. Values are: 0 = clear moonlight mode 1 = set moonlight mode	LifeView FlyCam CF

Option	Setting	Example of supported device
CamOptPowerDown!	Turns off the device. Set the value to 1 to turn off the device.	HP PhotoSmart VEO 130S Hitachi G1000
CamOptPowerUp!	Turns on the device. Set the value to 1 to turn on the device.	Hitachi G1000
CamOptPreviewPosLeft!	The left-side position of the preview area. The preview area's size is fixed at 160 x 120 pixels.	Hitachi G1000
CamOptPreviewPosTop!	The top-side position of the preview area. The preview area's size is fixed at 160 x 120 pixels.	Hitachi G1000
CamOptQuality!	Picture quality. This option determines the level of compression. The greater the compression, the lower the picture quality. Values are: 0 = good 1 = better 2 = best	HP PhotoSmart VEO 130S
CamOptSaturation!	Amount of color saturation (relative purity of color). The value is an integer in the range 0 to 1000.	LifeView FlyCam CF
CamOptSharpen!	Adjusts sharpness of the image. The value is an integer in the range 1 to 100. For example: 0 = soft (blurred) edges 50 = clear (default) 100 = sharp edges	LifeView FlyCam CF
CamOptTimeOut!	Timeout in seconds for any capture process. Used by devices that can capture an image asynchronously to determine how long the capture function waits for a response. This value is ignored for synchronous devices.	
CamOptWhiteBalance!	Adjusts the image to the current light conditions. Values are: 0 = Automatic White Balance (AWD) 1 = Sun/Daylight 2 = Tungsten/Incandescent 3 = Fluorescent 4 = "effects" 5 = Dim/Cloudy	HP PhotoSmart VEO 130S LifeView FlyCam CF Hitachi G1000

Examples

The following statements get the value of the CamOptWhiteBalance option:

```
integer li_return
li_return = g_myCamera.GetOption(CamOptWhiteBalance)
if not li_return = 0 then
   sle_opt.text = "White Balance: " + string(li_return)
```

```
else
   sle_opt.text = "White Balance: Unsupported option."
end if
```

See also

CaptureImage
HasOption
Open
SetOption

GetOrigin

Description

Finds the X and Y coordinates of the upper-left corner of the ListView item.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

ListView controls

Syntax

listviewname.**GetOrigin** (*x* , *y*)

Argument	Description
listviewname	The ListView control for which you want to find the coordinates of the upper-left corner
x	An integer variable in which you want to store the X coordinate for the ListView control
y	An integer variable in which you want to store the Y coordinate for the ListView control

Return value

Integer. Returns 1 if it succeeds and – 1 if it fails.

Usage

Use GetOrigin to find the position of a dragged object relative to the upper left corner of a ListView control.

Examples

This example moves a static text clock to the upper-left coordinates of the selected ListView item:

```
integer li_index
listviewitem l_lvi

li_index = lv_list.SelectedIndex()
lv_list.GetItem(li_index, l_lvi)
lv_list.GetOrigin(l_lvi.ItemX, l_lvi.ItemY)

sle_info.Text = "X is "+ String(l_lvi.ItemX) &
   + " and Y is " + String(l_lvi.ItemY)

st_clock.Move(l_lvi.itemx , l_lvi.ItemY)

MessageBox("Clock Location", "X is " &
   + String(st_clock.X) &
   + ", and Y is " &
   + String(st_clock.Y)+".")
```

GetParagraphSetting

Description

Gets the size of the indentation, left margin, or right margin of the paragraph containing the insertion point in a RichTextEdit control.

PocketBuilder	✗
PowerBuilder	✓

Applies to

RichTextEdit controls

Syntax

rtecontrol.**GetParagraphSetting** (*whichsetting*)

Return value

Long. Returns the size of the specified setting in thousandths of an inch. GetParagraphSetting returns -1 if an error occurs. If *whichsetting* is null, it returns null.

GetParent

Description

Obtains the parent of the specified object.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Any object

Syntax

objectname.**GetParent** ( )

Argument	Description
objectname	A control in a window or user object or an item on a menu for which you want the parent object

Return value

PowerObject. Returns a reference to the parent of *objectname*.

Examples

In event scripts for a user object that will be used as a tab page, you can use code like the following to make references to the parent Tab control generic:

```
// a_tab is generic;
// it does not know about specific pages
tab a_tab

// a_tab_page is generic;
// it does not know about specific controls
userobject a_tab_page

// Get values for the Tab control and the tab page
a_tab = this.GetParent( )
// Somewhat redundant, for illustration only
a_tab_page = this

// Set properties for the tab page
a_tab_page.PowerTipText = "Important property page"
// Set properties for the Tab control
a_tab.PowerTips = TRUE

// Run Tab control functions
a_tab.SelectTab(a_tab_page)
```

You cannot refer to controls on the user object because *a_tab_page* does not know about them. You cannot refer to specific pages in the Tab control because *a_tab* does not know about them either.

In event scripts for controls on the tab page user object, you can use two levels of GetParent to refer to the user object and the Tab control containing the user object as a tab page:

```
// For a control, add one more level of GetParent()
// and you can make the same settings as above
tab a_tab
userobject a_tab_page

a_tab_page = this.GetParent()
a_tab = a_tab_page.GetParent()

a_tab_page.PowerTipText = "Important property page"
```

```
a_tab.PowerTips = TRUE

a_tab.SelectTab(a_tab_page)
```

See also

ParentWindow
"Pronouns" on page 10

GetPin

Description

Called by EAServer to obtain a PIN for use with an SSL connection. This function is used by PowerBuilder clients connecting to EAServer.

PocketBuilder	✗
PowerBuilder	✓

Applies to

SSLCallBack objects

Syntax

sslcallback.**GetPin** (*thesessioninfo, timedout*)

Return value

String. Returns the PIN specified by the user.

GetRecipients

Description

Gets an array of recipients from the POOMRecipient object.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✗

Applies to

POOMAppointment objects

Syntax

Integer *objectname*.GetRecipients (*recipients[ ]*)

Argument	Description
objectname	The name of the POOMAppointment object
recipients[ ]	An array of POOMRecipient objects

Return value

Integer. Returns 1 for success and one of the following negative values if an error occurs:

-1	Unspecified error

-2	Cannot connect to the repository or a required internal subobject failed to connect to the repository
-3	Cannot log in to the repository
-4	Incorrect input argument
-5	Action cannot be performed
-6	The object identifier (OID) is not in the repository
-7	Feature is not implemented yet
-8	No matching entries found for the criteria

See also

AddRecipient
RemoveRecipient

GetRecordSet

Description

Returns the current ADO Recordset object.

PocketBuilder	✗
PowerBuilder	✓

Applies to

ADOResultSet objects

Syntax

adoresultset.**GetRecordSet** (*adorecordsetobject*)

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs.

GetRecurrence

Description

Returns the recurrence pattern for the appointment.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✗

Applies to

POOMAppointment, POOMTask objects

Syntax

POOMRecurrence *objectname*.GetRecurrence ( )

Argument	Description
objectname	The name of the POOMAppointment or POOMTask object

Return value

POOMRecurrence. Use IsValid to determine whether a valid POOMRecurrence was returned.

See also

ClearRecurrencePattern
SetRecurrence
SkipRecurrence

GetRemote

Asks a DDE server application to provide data and stores that data in the specified variable. There are two ways of calling GetRemote, depending on the type of DDE connection you have established.

To	Use
Make a single request of a DDE server application (called a cold link)	Syntax 1
Request data from a DDE server application after you have opened a channel (called a warm link)	Syntax 2

Syntax 1 For single DDE requests

Description

Asks a DDE server application to provide data and stores that data in the specified variable without requiring an open channel. This syntax is appropriate when you will make only one or two requests of the server.

PocketBuilder	✗
PowerBuilder	✓

Syntax

GetRemote (*location*, *target*, *applname*, *topicname*)

Return value

Integer. Returns 1 if it succeeds and a negative integer if an error occurs. Values are:

-1	Link was not started
-2	Request denied

Syntax 2 For DDE requests via an open channel

Description

Asks a DDE server application to provide data and stores that data in the specified variable when you have already established a warm link by opening a channel to the server. A warm link, with an open channel, is more efficient when you intend to make several DDE requests.

PocketBuilder	✗
PowerBuilder	✓

Syntax

GetRemote (*location*, *target*, *handle* {, *windowhandle* })

Return value

Integer. Returns 1 if it succeeds and a negative integer if an error occurs.

GetSatellitesInView

Description

Populates a GPSSatellitesInView structure with data about the satellites currently in view.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✗

Applies to

GPS and SerialGPS objects

Syntax

Integer *GPSname*.GetSatellitesInView(*GPSSatellitesInView*)

Argument	Description
GPSname	Name of the GPS or SerialGPS object
GPSSatellitesInView	Structure passed by reference that stores position information for the satellites in view

Return value

Integer. Returns 1 for success and 100 or a negative number for an error. The following is a list of possible error codes and their meanings:

100 End of buffer. The requested data was not found.

-1 General error.

-10 Invalid object. Could occur if the GPS object instance is corrupted.

-13 Not previously opened. This function cannot be called until a GPS object or SerialGPS object has been successfully opened.

-14 Read timeout. Occurs when the timeout interval (a ConfigParams property of the SerialGPS object) is exceeded.

-15 Read Failure. Unable to read the file or serial port.

-16 Parser Error. Parser is unable to interpret a sentence. This error is generated when nonstandard tokens are discovered while parsing the GPS data.

-17 Checksum Error. Most GPS sentences end in a two-digit checksum value. The PocketBuilder parser verifies this value and reports a checksum error if the calculated value does not match the stated value.

Usage

Use this function to populate a GPSSatellitesInView structure with information about the position of each satellite currently in view and the accuracy of the position fix. The HDOP (Horizontal Dilution of Precision) and VDOP (Vertical Dilution of Precision) properties indicate the level of confidence in the accuracy of measurements related to the horizontal and vertical positions of the satellites, based on current satellite geometry. A lower value indicates greater confidence.

Position information is returned in an array of GPSSatellitePosition structures, each of which contains information about the azimuth, elevation, signal strength, and PRN number of each of the satellites currently in view. Up to 12 satellites can be listed in the satellite array. Use the UpperBound function to determine the number of satellites listed.

Examples

The following lines create a SerialGPS object, retrieve information about the satellites used for the current position fix, test the validity of the GPSSatellitesInView object, and write data to a multiline edit box:

```
// instance variable: GPSSatellitePosition iGPS_SP[]

SerialGPS myGPS
GPSSatellitesInView mySIV
Integer rc

myGPS = CREATE SerialGPS
myGPS.open()
...
rc = myGPS.getSatellitesInView(mySIV)
if rc = 1 then
mle_1.text = "HDOP: " + String(mySiv.HDOP) + "~r~n"
mle_1.text += "VDOP: " + String(mySiv.VDOP) + "~r~n"
mle_1.text += "Satellites in view: PRN, Azimuth, " &
   + "Elevation, SNR values. ~r~n"

iGPS_SP = mySIV.Satellite[]
```

```
integer count
for count = 1 to UpperBound(iGPS_SP)
   mle_1.text += String(iGPS_SP[i].PRN + ", "
   mle_1.text += String(iGPS_SP[i].Azimuth + ", "
   mle_1.text += String(iGPS_SP[i].Elevation + ", "
   mle_1.text += String(iGPS_SP[i].SNR + "~r~n "
end for
else
//process error message
end if
```

See also

GetFix
GetHeading
Open

GetScreenOrientation

Description

Gets the screen orientation of a device or emulator capable of screen rotation.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✗

Syntax

Integer GetScreenOrientation ( )

Return value

The return values can be one of the following or a negative number for failure:

- **0** 0 degrees (the native orientation for the device)
- **1** 90 degrees (right-handed landscape orientation)
- **2** 180 degrees (upside down)
- **4** 270 degrees (left-handed landscape orientation)

Usage

This function is supported on devices using the Windows Mobile 2003 SE or later platform.

Examples

The following lines place the value for the current screen orientation in the lb_result list box:

```
integer iRet
iRet = GetScreenOrientation()
lb_result.AddItem( "Current Orientation: " &
      + string(iRet) )
```

See also SetScreenOrientation

GetSeriesStyle

Finds out the appearance of a series in a graph. The appearance settings for individual data points can override the series settings, so the values obtained from GetSeriesStyle may not reflect the current state of the graph. There are several syntaxes, depending on what settings you want.

To	Use
Get the series' colors	Syntax 1
Get the line style and width used by the series	Syntax 2
Get the fill pattern or symbol for the series	Syntax 3
Find out if the series is an overlay (a series shown as a line on top of another graph type)	Syntax 4

GetSeriesStyle provides information about a series. The data points in the series can have their own style settings. Use SetSeriesStyle to change the style values for a series. Use GetDataStyle to get style information for a data point and SetDataStyle to override series settings and set style information for individual data points.

The graph stores style information for properties that do not apply to the current graph type. For example, you can find out the fill pattern for a data point or a series in a two-dimensional line graph, but that fill pattern will not be visible.

Syntax 1 For the colors of a series

Description Obtains the colors associated with a series in a graph.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to Graph controls in windows and user objects, and graphs in DataWindow controls

Syntax

controlname.**GetSeriesStyle** ({ *graphcontrol*, } *seriesname*, *colortype*, *colorvariable*)

Argument	Description
controlname	The name of the graph in which you want to obtain the color of a series, or the name of the DataWindow control containing the graph.
graphcontrol (DataWindow control only)	(Optional) A string whose value is the name of the graph in the DataWindow control for which you want the color of a series.
seriesname	A string whose value is the name of the series for which you want the color.
colortype	A value of the grColorType enumerated datatype specifying the aspect of the series for which you want the color: • Foreground! — Text color • Background! — Background color • LineColor! — Line color • Shade! — Shade (for graphs that are 3-dimensional or have solid data markers)
colorvariable	A long variable in which you want to store the color's RGB value.

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs. Stores in *colorvariable* the RGB value of the specified series and item. If any argument's value is null, GetSeriesStyle returns null.

Examples

These statements store in the variable color_nbr the text (foreground) color used for a series in the graph gr_emp_data. The series name is the text in the SingleLineEdit sle_series:

```
long color_nbr
gr_emp_data.GetSeriesStyle(sle_series.Text, &
   Foreground!, color_nbr)
```

These statements store in the variable *color_nbr* the background color used for the series PCs in the graph gr_computers in the DataWindow control dw_equipment:

```
long color_nbr
// Get the color.
dw_equipment.GetSeriesStyle("gr_computers", &
   "PCs", Background!, color_nbr)
```

These statements store the color for the series under the mouse pointer in the graph gr_product_data in *line_color*:

```
string SeriesName
integer SeriesNbr, Data_Point
long line_color
grObjectType MouseHit

MouseHit = ObjectAtPointer(SeriesNbr, Data_Point)

IF MouseHit = TypeSeries! THEN
   SeriesName = &
      gr_product_data.SeriesName(SeriesNbr)

   gr_product_data.GetSeriesStyle(SeriesName, &
      LineColor!, line_color)
END IF
```

See also

AddSeries
GetDataStyle
FindSeries
SetDataStyle
SetSeriesStyle

Syntax 2 For the line style and width used by a series

Description

Obtains the line style and width for a series in a graph.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Graph controls in windows and user objects, and graphs in DataWindow controls

Syntax

controlname.**GetSeriesStyle** ({ *graphcontrol*, } *seriesname*, *linestyle*, *linewidth*)

Argument	Description
controlname	The name of the graph for which you want the line style and width for a series in a graph, or the name of the DataWindow control containing the graph.
graphcontrol (DataWindow control only)	(Optional) A string whose value is the name of the graph in the DataWindow control for which you want the line style information.
seriesname	A string whose value is the name of the series for which you want the line style information.

Argument	Description
linestyle	A variable of type LineStyle in which you want to store the line style of *seriesname*.
linewidth	An integer variable in which you want to store the line width for *seriesname*. The width is measured in pixels.

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs. Stores in *linestyle* a value of the LineStyle enumerated datatype and in *linewidth* the width of the line used for the specified series. If any argument's value is null, GetSeriesStyle returns null.

Examples

These statements store in the variables *line_style* and *line_width* the line style and width for the series under the mouse pointer in the graph gr_product_data:

```
string SeriesName
integer SeriesNbr, Data_Point, line_width
LineStyle line_style
grObjectType MouseHit

MouseHit = ObjectAtPointer(SeriesNbr, Data_Point)

IF MouseHit = TypeSeries! THEN
   SeriesName = &
      gr_product_data.SeriesName(SeriesNbr)

   gr_product_data.GetSeriesStyle(SeriesName, &
      line_style, line_width)
END IF
```

See also

AddSeries
GetDataStyle
FindSeries
SetDataStyle
SetSeriesStyle

Syntax 3 For the fill pattern or symbol of a series

Description

Obtains the fill pattern or symbol of a series in a graph.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Graph controls in windows and user objects, and graphs in DataWindow controls

Syntax

controlname.**GetSeriesStyle** ({ *graphcontrol*, } *seriesname*, *enumvariable*)

Argument	Description
controlname	The name of the graph for which you want the style information for a series in a graph, or the name of the DataWindow control containing the graph.
graphcontrol (DataWindow control only)	(Optional) A string whose value is the name of the graph in the DataWindow control for which you want the style information.
seriesname	A string whose value is the name of the series for which you want the style information.
enumvariable	The variable in which you want to store the style information. You can specify a FillPattern or grSymbolType variable. The style information that GetSeriesStyle stores depends on the variable type.

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs. Stores in *enumvariable* a value of the appropriate enumerated datatype for the fill pattern or symbol used for the specified series. If any argument's value is null, GetSeriesStyle returns null.

Usage

See SetSeriesStyle for a list of the enumerated datatype values that GetSeriesStyle stores in *enumvariable*.

Examples

These statements store in the variable *data_pattern* the fill pattern for the series under the mouse pointer in the graph gr_product_data:

```
string SeriesName
integer SeriesNbr, Data_Point
FillPattern data_pattern
grObjectType MouseHit

MouseHit = ObjectAtPointer(SeriesNbr, Data_Point)

IF MouseHit = TypeSeries! THEN
   SeriesName = &
      gr_product_data.SeriesName(SeriesNbr)

   gr_product_data.GetSeriesStyle(SeriesName, &
      data_pattern)
END IF
```

This example stores in the variable *data_pattern* the fill pattern for the series under the pointer in the graph gr_depts in the DataWindow control dw_employees. It then sets the fill pattern for the series Total Salary in the graph gr_dept_data to that pattern:

```
string SeriesName
integer SeriesNbr, Data_Point
FillPattern data_pattern
grObjectType MouseHit

MouseHit = &
   ObjectAtPointer("gr_depts" , SeriesNbr, &
      Data_Point)

IF MouseHit = TypeSeries! THEN
   SeriesName = &
       dw_employees.SeriesName("gr_depts" , SeriesNbr)

   dw_employees.GetSeriesStyle("gr_depts" , &
      SeriesName, data_pattern)

   gr_dept_data.SetSeriesStyle("Total Salary", &
      data_pattern)
END IF
```

In these examples, you can change the datatype of *data_pattern* (the variable specified as the last argument) to find out the symbol type.

See also

AddSeries
GetDataStyle
FindSeries
SetDataStyle
SetSeriesStyle

Syntax 4 For determining whether a series is an overlay

Description

Reports whether a series in a graph is an overlay—whether it is shown as a line on top of another graph type.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Graph controls in windows and user objects, and graphs in DataWindow controls

Syntax

controlname.**GetSeriesStyle** ({ *graphcontrol*, } *seriesname*,*overlayindicator*)

Argument	Description
controlname	The name of the graph for which you want the overlay status of a series in a graph, or the name of the DataWindow control containing the graph.
graphcontrol (DataWindow control only)	(Optional) A string whose value is the name of the graph in the DataWindow control for which you want the overlay status.
seriesname	A string whose value is the name of the series for which you want the overlay status.
overlayindicator	A boolean variable in which you want to store a value indicating whether the series is an overlay. GetSeriesStyle sets *overlayindicator* to true if the series is an overlay and false if it is not.

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs. Stores in *overlayindicator* true if the specified series is an overlay and false if it is not. If any argument's value is null, GetSeriesStyle returns null.

Examples

These statements find out whether a series in the graph gr_emp_data is an overlay. The series name is the text in the SingleLineEdit sle_series:

```
boolean is_overlay
gr_emp_data.GetSeriesStyle(sle_series.Text, &
   is_overlay)
```

GetShortName

Description

Gets the short name for the current PocketBuilder execution context.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to	ContextInformation objects
Syntax	*servicereference*.**GetShortName** (*shortname*)

Argument	Description
servicereference	Reference to the ContextInformation service instance.
shortname	String into which the function places the short name. This argument is passed by reference.

Return value

Integer. Returns 1 if the function succeeds and -1 if an error occurs. The function returns values for its *shortname* argument as follows:

- **PocketBuilder runtime on Pocket PC** PocketPC
- **PocketBuilder runtime on Pocket PC Phone Edition** PocketPC
- **PocketBuilder runtime on Smartphone** PocketSM
- **PowerBuilder runtime** PBRun
- **PowerBuilder window plug-in** PBWinPlugin
- **PowerBuilder window ActiveX** PBRTX

Usage

Call this function to determine the current execution environment.

Examples

This example calls the GetShortName function. *ci* is an instance variable of type ContextInformation:

```
String ls_name

this.GetContextService("ContextInformation", ci)
ci.GetShortName(ls_name)
IF ls_name <> "PBRun" THEN
   cb_close.visible = FALSE
END IF
```

See also

GetContextService
GetFixesVersion
GetHostObject
GetMajorVersion
GetMinorVersion
GetName
GetVersionName

GetSIPRect

Description

Gets the rectangular coordinates of the SIP, whether it is visible or not, on a Pocket PC device or emulator.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✗

Syntax

int GetDeskRect (long *left*, long *top*, long *right*, long *bottom*)

Argument	Description
left	Leftmost value of the rectangular area occupied by the SIP
top	Topmost value of the rectangular area occupied by the SIP
right	Rightmost value of the rectangular area occupied by the SIP
bottom	Bottommost value of the rectangular area occupied by the SIP

Return value

Integer. Returns 1 for success and -1 for failure.

Usage

Typically it is useful to know the topmost coordinate of the SIP so you can adjust the positions and sizes of controls on the current window when the SIP is visible.

On the desktop, GetSIPRect parameters *left*, *top*, *right*, and *bottom* always have the values 0, 214, 240, and 290, which are the coordinates for a typical SIP window on a Pocket PC device.

Examples

The following example displays the coordinates for the SIP in a multiline edit text box:

```
String strDisplay=""
int rc
long left = 0, top = 0, right = 0, bottom = 0

rc = GetSIPRect(left, top, right, bottom)
strDisplay +=("Desk RECT:~r~n~t Left = " +string(left)&
 +"~r~n~t Top=" + String(top) + "~r~n~t Right = " &
 + String(right)+ "~r~n~t Bottom = " + String(bottom))
mle_1.text = strDisplay
```

See also

GetDeskRect
IsSIPVisible

GetSIPType

Description

Returns the type of the current SIP window, whether it is visible or not.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✗

Syntax

SIPIMType GetSIPType ( )

Argument	Description
SIPIMType	An enumerated representation of the type of SIP. Supported values are: SIPKeyboard!, SIPBlock!, SIPJot!, SIPWordLogic!, SIPTranscriber!, and SIPFitaly!

Return value

SIPIMType. Returns -1 for failure. On the desktop, always returns SIPKeyboard! for the *SIPIMType*.

Usage

You can use this method to report the current SIP input method available to the application user.

Examples

The following example displays the type of the current SIP window in a multiline edit text box:

```
String strDisplay=""
SIPIMType sType
sType = GetSIPType()
choose case sType
   case SIPKeyboard!
      strDisplay += ("SIP TYPE IS Keyboard! ~r~n")
   case SIPJot!
      strDisplay += ("SIP TYPE IS SIPJot! ~r~n")
   case SIPBlock!
      strDisplay += ("SIP TYPE IS SIPBlock! ~r~n")
   case SIPWordLogic!
      strDisplay += ("SIP TYPE IS SIPWordLogic! ~r~n")
   case SIPTranscriber!
      strDisplay +=("SIP TYPE IS SIPTranscriber! ~r~n")
    case SIPFitaly!
      strDisplay +=("SIP TYPE IS Fitaly keyboard ~r~n")
    case else
      strDisplay+= ("ERROR!!! INVALID SIP TYPE ~r~n");
end choose
mle_1.text = strDisplay
```

See also

GetSIPRect
IsSIPVisible
SetSIPType

GetSpecialFolder

Description

Retrieves the name of a localized folder.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✗

Syntax

String GetSpecialFolder (*id_as_integer*)

Argument	Description
id_as_integer	Integer that corresponds to a localized system folder name on the desktop, or on a device or emulator. Table 10-5 in the Usage section displays the correlation between integer values and folder names.

Return value

String. Returns the name of the folder on a localized operating system.

Usage

Operating systems using different languages can have different names for system folders. The GetSpecialFolder system function provides a means of obtaining the name of a system folder on a specific desktop machine or Windows CE device. This enables you to develop applications that access system folders and can still be deployed to devices using different language operating systems.

You use the integer parameter of GetSpecialFolder to indicate the name of the system folder that you want to return. The *userName* variable in Table 10-5 depends on the login name of the current user.

Table 10-5: Correspondence of parameter value to folder name

Value	Desktop folder	Pocket PC folder	Smartphone folder
0	C:\Documents and Settings\ *userName*\Desktop	\My Documents	—
2	C:\Documents and Settings\ *userName*\Start Menu\Programs	\Windows\Start Menu\Programs	—
5	C:\Documents and Settings\ *userName*\My Documents	\My Documents	\Storage\My Documents

Value	Desktop folder	Pocket PC folder	Smartphone folder
6	C:\Documents and Settings\ *userName*\Favorites	\Windows\Favorites	\Storage\Windows\Favorites
7	C:\Documents and Settings\ *userName*\Start Menu\Programs\Startup	\Windows\Startup	\Storage\Windows\Startup
11	C:\Documents and Settings\ *userName*\Start Menu	\Windows\Start Menu	\Storage\Windows\Start Menu
20	C:\WINNT\Fonts	\Windows\Fonts	\Storage\Windows\Fonts
26	C:\Documents and Settings\ *userName*\Application Data	—	\Storage\Application Data
36	C:\WINNT\	Pocket PC 2002: — Pocket PC 2003 and later: \Windows	\Storage\Windows
38	C:\Program Files	\Program Files	\Storage\Program Files

Examples

The following example returns the name of the localized folder that corresponds to the \Windows\StartMenu\Programs folder on an English language Pocket PC device or emulator. For a German language Pocket PC device, the return value would be \Windows\Startmenü\Programme:

```
String ls_Folder
ls_Folder = GetSpecialFolder(2)
```

GetSpacing

Description

Obtains the line spacing of the paragraph containing the insertion point in a RichTextEdit control.

PocketBuilder	✗
PowerBuilder	✓

Applies to

RichTextEdit controls

Syntax

rtename.**GetSpacing** ( )

Return value

Spacing. A value of the Spacing enumerated datatype indicating the line spacing of the paragraph containing the insertion point.

GetStatus

Description	Returns the status of the EAServer transaction associated with the calling thread.

PocketBuilder	✗
PowerBuilder	✓

Applies to	CORBACurrent objects
Syntax	*CORBACurrent*.**GetStatus** ( )
Return value	Integer. Returns -1 if an error occurs and a positive integer if it succeeds.

GetSyncRegistryProperties

Description	Reserved for future use. Returns an integer to determine whether to use synchronization properties saved in the registry.
Applies to	MLSyncrhonization, MLSync controls
Syntax	*syncObject*.**GetSyncRegistryProperties** ( )

Argument	Description
syncObject	The name of the synchronization object.

Return value	Integer. Returns 1 for success and -1 for failure. Failure occurs if SyncRegistryKey property is not set or if the key does not exist in the Windows registry.

GetSupportedDecoders

Description	Retrieves the list of supported decoders.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✗

Applies to	BarcodeScanner objects

Syntax

Integer *scanner.*GetSupportedDecoders (*intDecoders* [ ])

Argument	Description
scanner	The scanner object that is associated with the scanner for which you want to obtain a list of enabled decoders
intDecoders [ ]	Array of integers that correspond to the decoder IDs of the supported decoders

Return value

Integer. Returns 1 for success or one of the following negative values if an error occurs:

- **-1** Unspecified error
- **-2** Supporting DLL not loaded error
- **-3** Initialization error other than DLL not loaded
- **-4** Error in the passed in arguments
- **-5** Something in the object instance is inconsistent
- **-6** Call to the driver failed
- **-7** Error opening the specific scan device
- **-8** Error in the internal buffer allocation
- **-9** Incorrect scan state for the requested action (typically benign)
- **-10** Low level device error
- **-100** Feature not implemented

Usage

The supported decoders are defined by the firmware of the scanner device. The subset of decoders to use for a scanning operation can be obtained by a GetEnabledDecoders function call.

Examples

The following example places the IDs of supported decoders in an array:

```
integer li_rtn, li_firstID, li_secondID, l_IDs[ ]
li_rtn = l_scanner.GetSupportedDecoders(l_IDs)
li_firstID = l_IDs[1]
li_secondID = l_IDs[2]
```

See also

EnableDecoder
GetEnabledDecoders

GetTask

Description

Gets a task from Pocket Outlook.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✗

Applies to

POOM objects

Syntax

POOMTask *objectname*.GetTask (*index*)

Argument	Description
objectname	The name of the POOM object
index	Integer for the index of the task that you want to retrieve

Return value

POOMTask. Use the IsValid function to confirm that a valid task was returned.

Usage

A user must be logged in to a POOM object to get a task from Pocket Outlook.

Examples

The following example retrieves the first task in Pocket Outlook and displays it:

```
POOMTask task
DateTime  dt

task = g_poom.GetTask(1)
if IsValid(task) then
   task.Display()
end if
```

See also

GetTaskFromOID
GetTasks

GetTaskFromOID

Description

Gets a task from Pocket Outlook using the object ID.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✗

Applies to

POOM objects

Syntax

POOMTask *objectname*.GetTaskFromOID (*oid*)

Argument	Description
objectname	The name of the POOM object
oid	Unsignedlong for the object ID of the task that you want to retrieve

Return value

POOMTask. Use IsValid to determine whether a valid task was returned.

Usage

A user must be logged in to a POOM object to get a task from Pocket Outlook.

Examples

The following example retrieves a task with an object ID of 12:

```
myTask = g_poom.getTaskFromOID(12)
if IsValid(myTask) then
   // Use myTask
end if
```

See also

GetContact
GetContacts

GetTasks

Description

Gets an array of tasks from Pocket Outlook after optionally filtering the array for matching criteria.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✗

Applies to

POOM objects

Syntax

Integer *objectname*.GetTasks ({*matchcriteria*,} *contacts* [ ])

Argument	Description
objectname	The name of the POOM object
matchcriteria	A string describing criteria you want to use to filter the list of tasks
contacts	An array of POOMTasks passed by reference

Return value

Integer. Returns 1 for success and one of the following negative values if an error occurs:

-1	Unspecified error

-2	Cannot connect to the repository or a required internal subobject failed to connect to the repository
-3	Cannot log in to the repository
-4	Incorrect input argument
-5	Action cannot be performed
-6	The object identifier (OID) is not in the repository
-7	Feature is not implemented yet
-8	No matching entries found for the criteria

Usage

A user must be logged in to a POOM object to get tasks from Pocket Outlook.

Examples

The following example retrieves tasks that match the criterion that the priority be high:

```
li_rtn = g_poom.getTasks &
    ("[Importance]='ImportanceHigh!'", myTasks [] )
```

The following example retrieves all the tasks in the list and writes the subject and start date of each task to a list box:

```
integer li_rc
POOMTask taskArray[]
POOMTask task
DateTime  dt
int idx

li_rc = g_poom.GetTasks( taskArray )

FOR idx=1 to UPPERBOUND(taskArray)
   task = taskArray[idx]
   lb_res.AddItem( "Subject: " + task.Subject )
   lb_res.AddItem( "Starts: " + task.StartDate )
NEXT

lb_res.SelectItem( lb_res.TotalItems() )
```

See also

GetTask
GetTaskFromOID

GetTextColor

Description

Obtains the color of selected text in a RichTextEdit control.

PocketBuilder	✗
PowerBuilder	✓

Applies to

RichTextEdit controls

Syntax

rtename.**GetTextColor** ( )

Return value

Long. Returns the long value that specifies the color of the currently selected text. If text of different colors is selected, GetTextColor returns the color of the first selected character. GetTextColor returns -1 if an error occurs.

GetTextStyle

Description

Finds out whether selected text has text styles (such as bold or italic) assigned to it.

PocketBuilder	✗
PowerBuilder	✓

Applies to

RichTextEdit controls

Syntax

rtename.**GetTextStyle** (*textstyle*)

Return value

Boolean. Returns true if the selected text is formatted with the specified text style and false if it is not. If *textstyle* is null, GetTextStyle returns null.

GetToolbar

Description

Gets the current values for alignment, visibility, and title of the specified toolbar.

PocketBuilder	✗
PowerBuilder	✓

Applies to

MDI frame and sheet windows

Syntax

window.**GetToolbar** (*toolbarindex*, *visible* {, *alignment* {, *floatingtitle* } })

Return value	Integer. Returns 1 if it succeeds. GetToolbar returns -1 if there is no toolbar for the index you specify or if an error occurs. If any argument's value is null, returns null.

GetToolbarPos

Gets position information for the specified toolbar.

To get	Use
Docking position of a docked toolbar	Syntax 1
Coordinates and size of a floating toolbar	Syntax 2

Syntax 1 For docked toolbars

Description: Gets the position of a docked toolbar.

PocketBuilder	✗
PowerBuilder	✓

Applies to: MDI frame and sheet windows

Syntax: *window*.**GetToolbarPos** (*toolbarindex*, *dockrow*, *offset*)

Return value: Integer. Returns 1 if it succeeds. GetToolbarPos returns -1 if there is no toolbar for the index you specify or if an error occurs. If any argument's value is null, GetToolbarPos returns null.

Syntax 2 For floating toolbars

Description: Gets the position and size of a floating toolbar.

PocketBuilder	✗
PowerBuilder	✓

Applies to: MDI frame and sheet windows

Syntax: *window*.**GetToolbarPos** (*toolbarindex*, *x*, *y*, *width*, *height*)

Return value — Integer. Returns 1 if it succeeds. GetToolbarPos returns -1 if there is no toolbar for the index you specify or if an error occurs. If any argument's value is null, returns null.

GetTransactionName

Description — Returns a string describing the EAServer transaction associated with the calling thread.

PocketBuilder	✗
PowerBuilder	✓

Applies to — CORBACurrent objects

Syntax — *CORBACurrent*.**GetTransactionName** ( )

Return value — String. Returns a printable string describing the transaction if a transaction exists and an empty string otherwise.

GetURL

Description — Returns HTML for the specified URL.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to — Inet objects

Syntax — *servicereference*.**GetURL** (*urlname*, *data*)

Return value — Integer. Returns 1 for success and a negative value if an error occurs. Possible values are:

-1	General error
-2	Invalid URL
-4	Cannot connect to the Internet

Usage — Call this function to access HTML source for a URL.

Data references a standard class user object that descends from InternetResult and that has an overridden InternetData function. This overridden function then performs the processing you want with the returned HTML. Because the Internet returns data asynchronously, *data* must reference a variable that remains in scope after the function executes (such as a window-level instance variable).

For more information on the InternetResult standard class user object and the InternetData function, use the PocketBuilder Browser.

Examples

This example calls the GetURL function. *Iinet_base* is an instance variable of type inet:

```
iir_msgbox = CREATE n_ir_msgbox
iinet_base.GetURL(sle_url.text, iir_msgbox)
```

See also

HyperLinkToURL
InternetData
PostURL

GetVersionName

Description

Gets complete version information for the current PocketBuilder execution context. A complete version includes a major version, a minor version, and a fix level (such as 1.5.2).

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

ContextInformation objects

Syntax

servicereference.**GetVersionName** (*name*)

Argument	Description
servicereference	Reference to the ContextInformation service instance.
name	String into which the function places the version name. This argument is passed by reference.

Return value

Integer. Returns 1 if the function succeeds and -1 if an error occurs.

Usage

Call this function to determine the maintenance level of the current context.

Examples

This example calls the GetVersionName function. *ci* is an instance variable of type ContextInformation:

```
String ls_name
String ls_version
Constant String ls_currver = "8.0.3"

GetContextService("ContextInformation", ci)
ci.GetVersionName(ls_version)
IF ls_version <> ls_currver THEN
   MessageBox("Error", &
      "Must be at Version " + ls_currver)
END IF
```

See also

GetCompanyName
GetFixesVersion
GetHostObject
GetMajorVersion
GetMinorVersion
GetName
GetShortName

Handle

Description

Obtains the Windows handle of a PocketBuilder object. You can get the handle of the application, a window, or a control, but not a drawing object.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

Handle (*objectname* {, *previous* })

Argument	Description
objectname	The name of the object for which you want the handle. *Objectname* can be any PocketBuilder object, including an application or control, but cannot be a drawing object.
previous (optional)	(Obsolete argument) A boolean indicating whether you want the handle of the previous instance of an application. You can use this argument with the Application object only. In current versions of Windows, Handle always returns 0 when this argument is set to true.

Return value

Long. Returns the handle of *objectname*. If *objectname* is an application and *previous* is true, Handle always returns 0.

If *objectname* cannot be referenced at runtime, Handle returns 0 (for example, if *objectname* is a window and is not open).

Usage

Use Handle when you need an object handle as an argument to Windows Software Development Kit (SDK) functions or the PowerScript Send function.

Use IsValid instead of the Handle function to determine whether a window is open.

When you ask for the handle of the application, Handle returns 0 when you are using the PowerScript Run command. As far as Windows is concerned, your application does not have a handle when it is run from PocketBuilder. When you build and run an executable version of your application, the Handle function returns a valid handle for the application.

If you ask for the handle of a previous instance of an application by setting the previous flag to true, Handle always returns 0 in current versions of Windows. Use the Windows FindWindow function to determine whether an instance of the application's main window is already open.

Examples

This statement returns the handle to the window w_child:

```
Handle(w_child)
```

These statements use an external function called FlashWindow to change the title bar of a window to inactive and then return it to active. The external function declaration is:

```
function boolean flashwindow(uint hnd, boolean inst) &
   library "user.exe"
```

The code that flashes the window's title bar is:

```
integer nLoop   // Loop counter
long hWnd   // Handle to control

// Get the handle to a PowerBuilder window.
hWnd = Handle(Parent)
// Make the title bar flash 300 times.
FOR nLoop = 1 to 300
   FlashWindow (hWnd, true)
NEXT
// Return the window to its original state.
FlashWindow (hWnd, FALSE)
```

For applications, the Handle function does not return a useful value when the *previous* flag is true. You can use the FindWindow Windows function to determine whether a Windows application is already running. FindWindow returns the handle of a window with a given title.

Declare FindWindowA and SetForegroundWindow as global external functions:

```
PUBLIC FUNCTION unsignedlong FindWindow (long  &
   classname, string windowname) &
   LIBRARY "user32.dll" ALIAS FOR FindWindowA
PUBLIC FUNCTION int FindWindowA (long classname,  &
   string windowname) LIBRARY "user32.dll"
```

PocketBuilder applications
In PocketBuilder, you would declare two versions of each function, for use in testing on the desktop and in the deployed application:

```
public FUNCTION unsignedlong FindWindow_NT( long &
   ClassName, string WindowName )   &
   LIBRARY "user32.dll"  ALIAS FOR "FindWindowW"

public FUNCTION unsignedlong FindWindow_CE( long  &
   ClassName, string WindowName )   &
   LIBRARY "coredll.dll" ALIAS FOR "FindWindowW"

public FUNCTION int SetForegroundWindow_NT(   &
   unsignedlong hwnd ) LIBRARY "user32.dll"   &
   ALIAS FOR "SetForegroundWindow"

public FUNCTION int SetForegroundWindow_CE(  &
   unsignedlong hwnd ) LIBRARY "coredll.dll"   &
   ALIAS FOR "SetForegroundWindow"
```

Then add code like the following to your application’s Open event:

```
unsignedlong hwnd

hwnd = FindWindow( 0, "Main Window")
if hwnd = 0 then
   // no previous instance, so open the main window
   open( w_main )
else
   // open the previous instance window and halt
   SetForegroundWindow( hwnd )
   HALT CLOSE
end if
```

See also

Send

HasOption

Description

Determines whether the device supports a specific option.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✗

Applies to

Camera objects

Syntax

Boolean *objectname*.HasOption (*Opt*)

Argument	Description
objectname	The name of the camera object that you want to inquire about.
Opt	A value of the CameraOption enumerated variable that specifies the name of the option that you want to want to inquire about. For a list of options, see GetOption.

Return value

Integer. Returns 1 for success, and one of the following negative values if an error occurs:

-1	Unspecified error
-2	Supporting DLL not loaded
-3	Other initialization error
-5	Inconsistency in this object instance
-6	Call to the driver or device failed
-7	Unsupported option
-8	Value for option is out of range

Usage

Use the HasOption function to determine whether the camera device supports a specific option. You can call GetOption to return the option's value in a reference variable.

Examples

The following statements determine whether the device supports the CamOptWhiteBalance option and, if it does, use the GetOption function to return the value:

```
boolean lb_query
```

```
integer li_return
lb_query = g_myCam.HasOption(CamOptWhiteBalance)
if lb_query = true then
   li_return = g_myCam.GetOption(CamOptWhiteBalance)
end if
```

See also

CaptureImage
GetOption
Open
SetOption

Hide

Description

Makes an object or control invisible. Users cannot interact with an invisible object. It does not respond to any events, so the object is also, in effect, disabled.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Any object

Syntax

objectname.**Hide** ( )

Argument	Description
objectname	The name of the object or control you want to make invisible

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs. If *objectname* is null, Hide returns null.

Usage

If the object you want to hide is already invisible, then Hide has no effect.

You cannot use Hide to hide a drop-down or cascading menu or any menu that has an MDI frame window as its parent window. Nor can you hide a window that has been opened as an MDI sheet.

You can use the Disable function to disable menu items, which displays them in the disabled color and makes them inactive.

To disable an object so that it does not respond to events, but is still visible, set its Enabled property.

You can set an object's Visible property instead of calling Hide:

objectname.Visible = **false**

This statement:

```
lb_Options.Visible = FALSE
```

is equivalent to:

```
lb_Options.Hide()
```

Examples

This statement hides the ListBox lb_options:

```
lb_options.Hide()
```

In the script for a menu item, this statement hides the CommandButton cb_delete on the active sheet in the MDI frame w_mdi. The active sheets are of type w_sheet:

```
w_sheet w_active
w_active = w_mdi.GetActiveSheet()
IF IsValid(w_active) THEN w_active.cb_delete.Hide()
```

See also

Show

Hour

Description

Obtains the hour in a time value. The hour is based on a 24-hour clock.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

Hour (*time*)

Argument	Description
time	The time from which you want to obtain the hour

Return value

Integer. Returns an integer (00 to 23) whose value is the hour portion of *time*. If *time* is null, Hour returns null.

Examples

This statement returns the current hour:

```
Hour(Now())
```

This statement returns 19:

```
Hour(19:01:31)
```

See also

Minute
Now
Second
Hour method for DataWindows in the *DataWindow Reference*

HyperLinkToURL

Description

Opens the default Web browser, displaying the specified URL.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Inet objects

Syntax

servicereference.**HyperlinkToURL** (*url*)

Return value

Integer. Returns 1 if the function succeeds and -1 if an error occurs.

Usage

Call this function to display a URL from a PocketBuilder application.

Examples

This example calls the HyperlinkToURL function. *Iinet_base* is an instance variable of type inet:

```
GetContextService("Internet", iinet_base)
iinet_base.HyperlinkToURL(sle_url.text)
```

See also

GetURL
PostURL

Icon

Description

Specifies an icon to display in the notification tray when a notification event occurs.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✗

Applies to

NotificationBubble objects

Syntax

Integer *controlname*.Icon (*iconName*)

Argument	**Description**
controlname	The name of the notification bubble for which you want to assign an icon
iconName	String that contains the name of the icon you want to display for a notification event

Return value

Integer. Returns 1 for success, -1 if the *iconName* argument is empty, and -2 if there is an error retrieving the icon.

Usage

When a user taps the notification icon in the notification tray, a notification bubble displays on the Pocket PC device or emulator.

The notification icon should be an 8- or 16-bit 16x16 ICO file. It must be packaged into a resource file (PKR) that is deployed along with the application. The path to the icon listed in the PKR file can be relative to the directory for the application and PKR file, or it must match the exact path to the icon on the device or emulator.

Examples

The following example sets a 16x16 icon to display in the notification tray when a notification event occurs:

```
li_rtn = nb_myBubble.Icon("foo.ico")
```

See also

Remove
SetMessageSink

Idle

Description

Sets a timer so that PocketBuilder triggers an Application Idle event when there has been no user activity for a specified number of seconds.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

Idle (*n*)

Argument	Description
n	The number of seconds of user inactivity allowed before PocketBuilder triggers an Application Idle event. A value of 0 terminates Idle detection.

Return value

Integer. Returns 1 if it starts the timer, and -1 if it cannot start the timer or *n* is 0 and the timer has not been started. Note that when the timer has been started and you change *n*, Idle does not start a new timer; it resets the current timer interval to the new number of seconds. If *n* is null, Idle returns null. The return value is usually not used.

Usage

Use Idle to shut off or restart an application when there is no user activity. This is often done for security reasons.

Idle starts a timer after each user activity (such as a keystroke or a mouse click), and after *n* seconds of inactivity it triggers an Idle event. The Idle event script for an application typically closes some windows, logs off the database, and exits the application or calls the Restart function.

The timer is reset when any of the following activities occur:

- A mouse movement or mouse click in any window of the application
- Any keyboard activity when a window of the PowerBuilder application is current
- A mouse click or any mouse movement over the icon when a PowerBuilder application is minimized
- Any keyboard activity when the PowerBuilder application is minimized and is current (its name is highlighted)
- Any retrieval on a visible DataWindow that causes the edit control to be painted

Tip
To capture movement, write script in the MouseMove or Key events of the window or sheet. (Keyboard activity does not trigger MouseMove events.) Disable the DataWindow control and tab ordering during iterative retrieves so the Idle timer is not reset.

Examples

This statement sends an Idle event after five minutes of inactivity:

```
Idle(300)
```

This statement turns off idle detection:

```
Idle(0)
```

This example shows how to use the Idle event to stop the application and restart it after two minutes of inactivity. This is often used for computers that provide information in a public place.

Include this statement in the script for the application's Open event:

```
Idle(120) // Sends an Idle event after 2 minutes.
```

Include these statements in the script for the application's Idle event to terminate the application and then restart it:

```
// Statements to set the database to the desired
// state
...
Restart() // Restarts the application
```

See also

Restart
Timer

ImpersonateClient

Description

Allows a COM object running on MTS or COM+ to take on the security attributes of the client for the duration of a call.

PocketBuilder	✗
PowerBuilder	✓

Applies to

TransactionServer objects

Syntax

transactionserver.**ImpersonateClient** ( )

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs.

ImportClipboard

Description

Inserts data into a DataWindow control, DataStore object, or graph control from tab-separated, comma-separated, or XML data on the clipboard.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

XML data
XML data is not supported in this release of PocketBuilder.

For DataWindow and DataStore syntax, see the ImportClipboard method for DataWindows in the *DataWindow Reference* or the online Help.

Applies to

Graph controls in windows and user objects. Does not apply to graphs within DataWindow objects, because their data comes directly from the DataWindow.

Syntax

graphname.**ImportClipboard** ({ *importtype*}, { *startrow* {, *endrow* {, *startcolumn* } } })

Argument	Description
importtype (optional)	An enumerated value of the SaveAsType DataWindow constant. Valid type arguments for ImportClipboard are: Text! CSV! XML!
graphname	The name of the graph control to which you want to copy data from the clipboard.
startrow (optional)	The number of the first detail row in the clipboard that you want to copy. The default is 1. For default XML import, if *startrow* is supplied, the first *N* (*startrow* -1) elements are skipped, where *N* is the DataWindow row size. For template XML import, if *startrow* is supplied, the first (*startrow* -1) occurrences of the repetitive row mapping defined in the template are skipped.

Argument	Description
endrow (optional)	The number of the last detail row in the clipboard that you want to copy. The default is the rest of the rows. For default XML import, if *endrow* is supplied, import stops when *N* * *endrow* elements have been imported, where *N* is the DataWindow row size. For template XML import, if *endrow* is supplied, import stops after *endrow* occurrences of the repetitive row mapping defined in the template have been imported.
startcolumn (optional)	The number of the first column in the clipboard that you want to copy. The default is 1. For default XML import, if *startcolumn* is supplied, import skips the first (*startcolumn* - 1) elements in each row. This argument has no effect on template XML import.

Return value

Returns the number of rows that were imported if it succeeds and one of the following negative integers if an error occurs:

-1 No rows or *startrow* value supplied is greater than the number of rows in the string

-2 Input data does not match number of columns or required column type

-3 Invalid argument

-4 Invalid input

-11 XML Parsing Error; XML parser libraries not found, or XML not well formed

-12 XML Template does not exist or does not match the DataWindow

If any argument's value is null, ImportClipboard returns null. If the optional *importtype* argument is specified and is not a valid type, ImportClipboard returns -3.

Usage

The clipboard data must be formatted in tab-separated or comma-separated columns or in XML. The datatypes and order of the DataWindow object's columns must match the data on the clipboard.

For graphs, ImportClipboard uses only three columns and ignores other columns. Each row of data must contain three pieces of information. The information depends on the type of graph:

- For all graph types except scatter, the first column to be imported is the series name, the second column contains the category, and the third column contains the data.
- For scatter graphs, the first column to be imported is the series name, the second column is the data's x value, and the third column is the y value.

If a series or category already exists in the graph, the data is assigned to it. Otherwise, the series and categories are added to the graph.

You can add data to more than one series by specifying different series names in the first column.

Examples

If the clipboard contains the data shown below and the graph does not have any data yet, then the next statement produces a graph with two series and three categories. The clipboard data is:

```
Sales 94Jan3000
Sales 94Mar2200
Sales 94May2500
Sales 95Jan4000
Sales 95Mar3200
Sales 95May3500
```

This statement copies all the data in the clipboard, as shown above, to gr_employee:

```
gr_employee.ImportClipboard()
```

This statement copies the data from the clipboard starting with row 2 column 3 and copying to row 30 column 5 to the graph gr_employee:

```
gr_employee.ImportClipboard(2, 30, 3)
```

See also

ImportFile
ImportString

ImportFile

Description

Inserts data into a DataWindow control, DataStore object, or graph control from data in a file. The data can be tab-separated text, comma-separated text, or XML. The format of the file depends on whether the target is a DataWindow (or DataStore) or a graph and on the type of graph.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

XML files
XML data is not supported in this release of PocketBuilder.

For DataWindow and DataStore syntax, see the ImportFile method for DataWindows in the *DataWindow Reference* or the online Help.

Applies to

Graph controls in windows and user objects. Does not apply to graphs within DataWindow objects, because their data comes directly from the DataWindow.

Syntax

graphname.**ImportFile** ({ *importtype*}, *filename* {, *startrow* {, *endrow* {, *startcolumn* } } })

Argument	Description
graphname	The name of the graph control to which you want to copy data from the specified file.
importtype (optional)	An enumerated value of the SaveAsType DataWindow constant. If this argument is specified, the *importtype* argument can be specified without an extension. Valid type arguments for ImportFile are: Text! CSV! XML! (PowerBuilder only)
filename	A string whose value is the name of the file from which you want to copy data. The file must be a tab-separated file (TXT), comma-separated file (CSV), or Extensible Markup Language (XML). Specify the file's full name. If the optional *importtype* is not specified, the name must end in the appropriate extension. If you do not specify *filename* or if it is null, ImportFile prompts the user for a file name. The remaining arguments are ignored.

Argument	Description
startrow (optional)	The number of the first detail row in the file that you want to copy. The default is 1. For default XML import, if *startrow* is supplied, the first *N* (*startrow* -1) elements are skipped, where *N* is the DataWindow row size. For template XML import, if *startrow* is supplied, the first (*startrow* -1) occurrences of the repetitive row mapping defined in the template are skipped.
endrow (optional)	The number of the last detail row in the file that you want to copy. The default is the rest of the rows. For default XML import, if *endrow* is supplied, import stops when *N* * *endrow* elements have been imported, where *N* is the DataWindow row size. For template XML import, if *endrow* is supplied, import stops after *endrow* occurrences of the repetitive row mapping defined in the template have been imported.
startcolumn (optional)	The number of the first column in the file that you want to copy. The default is 1. For default XML import, if *startcolumn* is supplied, import skips the first (*startcolumn* - 1) elements in each row. This argument has no effect on template XML import.

Return value

Long. Returns the number of rows that were imported if it succeeds and one of the following negative integers if an error occurs:

-1	No rows or *startrow* value supplied is greater than the number of rows in the file
-2	Empty file or input data does not match number of columns or required column type
-3	Invalid argument
-4	Invalid input
-5	Could not open the file
-6	Could not close the file
-7	Error reading the text
-8	Unsupported file name suffix (must be *.txt, *.csv, *.dbf or *.xml)
-10	Unsupported dBase file format

-11 XML Parsing Error; XML parser libraries not found or XML not well formed

-12 XML Template does not exist or does not match the DataWindow

If any argument's value is null, ImportFile returns null. If the optional *importtype* argument is specified and is not a valid type, ImportFile returns -3.

Usage

The format of the file can be indicated by specifying the optional *importtype* parameter, or by including the appropriate file extension.

For graph controls, ImportFile only uses three columns and ignores other columns. Each row of data must contain three pieces of information. The information depends on the type of graph:

- For all graph types except scatter, the first column to be imported is the series name, the second column contains the category, and the third column contains the data.
- For scatter graphs, the first column to be imported is the series name, the second column is the data's x value, and the third column is the y value.

You can add data to more than one series by specifying different series names in the first column. To let users select the file to import, specify a null string for *filename*. PocketBuilder displays the Select Import File dialog box.

Double quotes The location and number of double quote marks in a field in a tab delimited file affect how they are handled when the file is imported. If a string is enclosed in one pair of double quotes, the quotes are discarded. If it is enclosed in three pairs of double quotes, one pair is retained when the string is imported. If the string is enclosed in two pairs of double quotes, the first pair is considered to enclose a null string, and the rest of the string is discarded.

When there is a double quote at the beginning of a string, any characters after the second double quote are discarded. If there is no second double quote, the tab character delimiting the fields is not recognized as a field separator and all characters up to the next occurrence of a double quote, including a carriage return, are considered to be part of the string. A validation error is generated if the combined strings exceed the length of the first string.

Double quotes after the first character in the string are rendered literally. Here are some examples of how tab-delimited strings are imported into a two-column DataWindow:

Text in file	Result
"Joe" TAB "Donaldson"	Joe Donaldson
Bernice TAB """Ramakrishnan"""	Bernice "Ramakrishnan"
""Mary"" TAB ""Li""	Empty cells

Text in file	**Result**
"Mich"ael TAB """Lopes"""	Mich "Lopes"
"Amy TAB Doherty"	Amy<TAB>Doherty in first cell, second cell empty
3""" TAB 4"	3""" 4"

Specifying a *null* string for file name
If you specify a null string for *filename*, the remaining arguments are ignored. All the rows and columns in the file are imported.

Examples

This statement copies all the data in the file *D:\EMPLOYEE.TXT* to gr_employee starting at the first row:

```
gr_employee.ImportFile("D:\EMPLOYEE.TXT")
```

This statement copies the data from the file *D:\EMPLOYEE.TXT* starting with row 2 column 3 and ending with row 30 column 5 to the graph gr_employee:

```
gr_employee.ImportFile("D:\EMPLOYEE.TXT", 2, 30, 3)
```

The following statements in a PowerBuilder application are equivalent. Both import the contents of the XML file named *myxmldata*:

```
gr_control.ImportFile(myxmldata.xml)
gr_control.ImportFile(XML!, myxmldata)
```

This example causes PocketBuilder to display the Specify Import File dialog box:

```
string null_str
SetNull(null_str)
dw_1.ImportFile(null_str)
```

See also

ImportClipboard
ImportString

ImportString

Description

Inserts data into a DataWindow control, DataStore object, or graph control from tab-separated, comma-separated, or XML data in a string. The way data is arranged in the string in tab-delimited columns depends on whether the target is a DataWindow (or DataStore) or a graph, and on the type of graph.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

XML data
XML data is not supported in this release of PocketBuilder.

For DataWindow and DataStore syntax, see the ImportString method for DataWindows in the *DataWindow Reference* or the online Help.

Applies to

Graph controls in windows and user objects. Does not apply to graphs within DataWindow objects, because their data comes directly from the DataWindow.

Syntax

graphname.**ImportString** (*{ importtype}, string* {, *startrow* {, *endrow* {, *startcolumn* } } })

Argument	Description
graphname	The name of the graph control to which you want to copy data from the specified string.
importtype (optional)	A value of the SaveAsType enumerated datatype (PocketBuilder or PowerBuilder) or a string (Web DataWindow) specifying the format of the imported string. If no import type is specified, the imported string should contain only tab-separated text. Valid type arguments are: Text! (default) CSV! XML!
string	A string from which you want to copy the data. The string should contain tab-separated or comma-separated columns or XML with one row per line (see Usage).

Argument	Description
startrow (optional)	The number of the first detail row in the string that you want to copy. The default is 1. For default XML import, if *startrow* is supplied, the first *N* (*startrow* -1) elements are skipped, where *N* is the DataWindow row size. For template XML import, if *startrow* is supplied, the first (*startrow* -1) occurrences of the repetitive row mapping defined in the template are skipped.
endrow (optional)	The number of the last detail row in the string that you want to copy. The default is the rest of the rows. For default XML import, if *endrow* is supplied, import stops when *N* * *endrow* elements have been imported, where *N* is the DataWindow row size. For template XML import, if *endrow* is supplied, import stops after *endrow* occurrences of the repetitive row mapping defined in the template have been imported.
startcolumn (optional)	The number of the first column in the string that you want to copy. The default is 1. For default XML import, if *startcolumn* is supplied, import skips the first (*startcolumn* - 1) elements in each row. This argument has no effect on template XML import.

Return value

Returns the number of data points that were imported if it succeeds and one of the following negative integers if an error occurs:

-1 No rows or *startrow* value supplied is greater than the number of rows in the string

-2 Empty string or input data does not match number of columns or required column type

-3 Invalid argument

-4 Invalid input

-11 XML Parsing Error; XML parser libraries not found or XML not well formed

-12 XML Template does not exist or does not match the DataWindow

If any argument's value is null, ImportString returns null. If the optional *importtype* argument is specified and is not a valid type, ImportString returns -3.

Usage

For graph controls, ImportString only uses three columns on each line and ignores other columns. The three columns must contain information that depends on the type of graph:

- For all graph types except scatter, the first column to be imported is the series name, the second column contains the category, and the third column contains the data.
- For scatter graphs, the first column to be imported is the series name, the second column is the data's x value, and the third column is the y value.

You can add data to more than one series by specifying different series names in the first column.

Examples

These statements copy the data from the string *ls_Text* starting with row 2 column 3 and ending with row 30 column 5 to the graph gr_employee:

```
string ls_Text
ls_Text = . . .
gr_employee.ImportString(ls_Text, 2, 30, 3)
```

The following script stores data for two series in the string *ls_gr* and imports the data into the graph gr_custbalance. The categories in the data are A, B, and C:

```
string ls_gr

ls_gr = "series1~tA~t12~r~n"
ls_gr = ls_gr + "series1~tB~t13~r~n"
ls_gr = ls_gr + "series1~tC~t14~r~n"
ls_gr = ls_gr + "series2~tA~t15~r~n"
ls_gr = ls_gr + "series2~tB~t14~r~n"
ls_gr = ls_gr + "series2~tC~t12.5~r~n"

gr_custbalance.ImportString(ls_gr, 1)
```

See also

ImportClipboard
ImportFile

IncomingCallList

Description

Provides a list of the callers of a routine included in a performance analysis model.

PocketBuilder on Desktop	✓
PocketBuilder on Pocket PC	✗
PocketBuilder on Smartphone	✗
PowerBuilder	✓

Applies to

ProfileRoutine object

Syntax

iinstancename.**IncomingCallList** (*list*, *aggregrateduplicateroutinecalls*)

Argument	**Description**
instancename	Instance name of the ProfileRoutine object.
list	An unbounded array variable of datatype ProfileCall in which IncomingCallList stores a ProfileCall object for each caller of the routine. This argument is passed by reference.
aggregateduplicateroutinecalls	A boolean indicating whether duplicate routine calls will result in the creation of a single or of multiple ProfileCall objects.

Return value

ErrorReturn. Returns one of the following values:

- Success!—The function succeeded
- ModelNotExistsError!—The model does not exist

Usage

Use this function to extract a list of the callers of a routine included in a performance analysis model. Each caller is defined as a ProfileCall object and provides the called routine and the calling routine, the number of times the call was made, and the elapsed time. The callers are listed in no particular order.

You must have previously created the performance analysis model from a trace file using the BuildModel function.

The *aggregateduplicateroutinecalls* argument indicates whether duplicate routine calls will result in the creation of a single or of multiple ProfileCall objects. This argument has no effect unless line tracing is enabled and a calling routine calls the current routine from more than one line. If *aggregateduplicateroutinecalls* is true, a new ProfileCall object is created that aggregates all calls from the calling routine to the current routine. If *aggregateduplicateroutinecalls* is false, multiple ProfileCall objects are returned, one for each line from which the calling routine called the called routine.

Examples

This example gets a list of the routines included in a performance analysis model and then gets a list of the routines that called each routine:

```
Long ll_cnt
ProfileCall lproc_call[]

lpro_model.BuildModel()
lpro_model.RoutineList(i_routinelist)

FOR ll_cnt = 1 TO UpperBound(iprort_list)
    iprort_list[ll_cnt].IncomingCallList(lproc_call, &
      TRUE)
     ...
NEXT
```

See also

BuildModel
OutgoingCallList

Init

Sets ORB property values or initializes an instance of the CORBACurrent service object.

To	Use
Set ORB property values for client connections to EAServer using the JaguarORB object	Syntax 1
Initialize an instance of the CORBACurrent service object for client- or component-managed transactions	Syntax 2

Syntax 1 For setting ORB property values

Description Sets ORB property values. This function is used by PowerBuilder clients connecting to EAServer.

PocketBuilder	✗
PowerBuilder	✓

Applies to JaguarORB objects

Syntax *jaguarorb*.**Init** (*options*)

Return value Long. Returns 0 if it succeeds and a negative number if an error occurs.

Syntax 2 For initializing CORBACurrent

Description Initializes an instance of the CORBACurrent service object.

PocketBuilder	✗
PowerBuilder	✓

Applies to CORBACurrent objects

Syntax *CORBACurrent*.**Init** ({ *connection* | *URL*})

Return value Integer. Returns 0 if it succeeds and a negative number if an error occurs.

InputFieldChangeData

Description Modifies the data value of input fields in a RichTextEdit control.

PocketBuilder	✗
PowerBuilder	✓

Applies to RichTextEdit controls

Syntax *rtename*.**InputFieldChangeData** (*inputfieldname*, *inputfieldvalue*)

Return value Integer. Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is null, InputFieldChangeData returns null.

InputFieldCurrentName

Description	Gets the name of the input field when the insertion point is in an input field in a RichTextEdit control.

PocketBuilder	✗
PowerBuilder	✓

Applies to	RichTextEdit controls
Syntax	*rtename*.**InputFieldCurrentName** ( )
Return value	String. Returns the name of the input field. If the insertion point is not in an input field or if an error occurs, it returns the empty string ("").

InputFieldDeleteCurrent

Description	Deletes the input field that is selected in a RichTextEdit control.

PocketBuilder	✗
PowerBuilder	✓

Applies to	RichTextEdit controls
Syntax	*rtename*.**InputFieldDeleteCurrent** ( )
Return value	Integer. Returns 1 if it succeeds and -1 if there is no input field at the insertion point, the input field is activated for editing, or an error occurs.

InputFieldGetData

Description	Get the data in the specified input field in a RichTextEdit control.

PocketBuilder	✗
PowerBuilder	✓

Applies to	RichTextEdit controls
Syntax	*rtename*.**InputFieldGetData** (*inputfieldname*)
Return value	String. The data in the input field. InputFieldGetData returns the empty string ("") if the field does not exist or an error occurs.

InputFieldInsert

Description	Inserts a named input field at the insertion point in a RichTextEdit control.

PocketBuilder	✗
PowerBuilder	✓

Applies to	RichTextEdit controls
Syntax	*rtename*.**InputFieldInsert** (*inputfieldname*)
Return value	Integer. Returns 1 if it succeeds and -1 if an error occurs. If *inputfieldname* is null, InputFieldInsert returns null.

InputFieldLocate

Description	Locates an input field in a RichTextEdit control and moves the insertion point there.

PocketBuilder	✗
PowerBuilder	✓

Applies to	RichTextEdit controls
Syntax	*rtename*.**InputFieldLocate** (*location* {, *inputfieldname* })
Return value	String. Returns the name of the input field it located if it succeeds. InputFieldLocate returns an empty string if no matching input field is found or if an error occurs. If any argument is null, InputFieldLocate returns null.

IsReadyToCapture

Description	Determines whether the device is ready to capture an image.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✗

Applies to	Camera objects

Syntax

objectname.IsReadyToCapture ( )

Argument	Description
objectname	The name of the camera object that you want to inquire about

Return value

Boolean. Returns true if the device is ready to capture an image and false otherwise.

Usage

Use the IsReadyToCapture function to determine whether the camera device is ready to capture an image.

Examples

The following statements determine whether the device is ready to capture an image and, if it is, use the CaptureImage function to capture the image:

```
boolean lb_query
integer li_return
lb_query = g_myCam.IsReadyToCapture()
if lb_query = true then
   li_return = g_myCam.CaptureImage("\myPic.jpeg")
end if
```

See also

CaptureImage
GetOption
Open
SetCaptureImageAttributes
SetOption

InsertCategory

Description

Inserts a category on the category axis of a graph at the specified position. Existing categories are renumbered to keep the category numbering sequential.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Graph controls in windows and user objects. Does not apply to graphs within DataWindow objects, because their data comes directly from the DataWindow.

Syntax

controlname.**InsertCategory** (*categoryvalue*, *categorynumber*)

Argument	Description
controlname	The name of the graph into which you want to insert a category.
categoryvalue	A value that is the category you want to insert. The category must be unique within the graph. The value you specify must be the same datatype as the datatype of the category axis.
categorynumber	The number of the category before which you want to insert the new category. To add the category at the end, specify 0. If the axis is sorted, the category will be integrated into the existing order, ignoring categorynumber.

Return value

Integer. Returns the number of the category if it succeeds and -1 if an error occurs. If the category already exists, it returns the number of the existing category. If any argument's value is null, InsertCategory returns null.

Usage

Categories are discrete. Even on a date or time axis, each category is separate with no timeline-style connection between categories. Only scatter graphs, which do not have discrete categories, have a continuous category axis.

When the axis datatype is string, category names are unique if they have different capitalization. Also, you can specify the empty string ("") as the category name. However, because category names must be unique, there can be only one category with that name.

When you use InsertCategory to create a new category, there will be holes in each of the series for that category. Use AddData or InsertData to create data points for the new category.

Equivalent syntax If you want to add a category to the end of a series, you can use AddCategory instead, which requires fewer arguments.

This statement:

```
gr_data.InsertCategory("Qty", 0)
```

is equivalent to:

```
gr_data.AddCategory("Qty")
```

Examples

These statements insert a category called Macs before the category named PCs in the graph gr_product_data:

```
integer CategoryNbr

// Get the number of the category.
CategoryNbr = FindCategory("PCs")
gr_product_data.InsertCategory("Macs", CategoryNbr)
```

In a graph reporting mail volume in the afternoon, these statements add three categories to a time axis. If the axis is sorted, the order in which you add the categories does not matter:

```
catnum = gr_mail.InsertCategory(13:00, 0)
catnum = gr_mail.InsertCategory(12:00, 0)
catnum = gr_mail.InsertCategory(13:00, 0)
```

See also

AddData
AddCategory
FindCategory
FindSeries
InsertData
InsertSeries

InsertClass

Description

Inserts a new object of the specified OLE class in an OLE control.

PocketBuilder	✗
PowerBuilder	✓

Syntax

ole2control.**InsertClass** (*classname*)

Return value

Integer. Returns 0 if it succeeds and a negative number if an error occurs.

InsertColumn

Description

Inserts a column with the specified label, alignment, and width at the specified location.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

ListView controls

Syntax

listviewname.**InsertColumn** (*index, label, alignment, width*)

Argument	Description
listviewname	The name of the ListView control to which you want to insert a column.
index	An integer whose value is the number of the column before which you are inserting a new column.
label	A string whose value is the name of the column you are inserting.
alignment	A value of the enumerated datatype Alignment specifying the alignment of the column you are inserting. Values are: Center! Justify! Left! Right!
width	An integer whose value is the width of the column you are inserting, in PowerBuilder units.

Return value

Integer. Returns the column *index* value if it succeeds and -1 if an error occurs.

Usage

You can insert a column anywhere in the control. If the index you specify is greater than the current number of columns, the column is inserted after the last column.

Examples

This example inserts a column named Location, makes it right-aligned, and sets the column width to 300:

```
lv_list.InsertColumn(2 , "Location" , Right! , 300)
```

See also

AddColumn
DeleteColumn

InsertData

Description

Inserts a data point in a series of a graph. You can specify the category for the data point or its position in the series. Does not apply to scatter graphs.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Graph controls in windows and user objects. Does not apply to graphs within DataWindow objects, because their data comes directly from the DataWindow.

Syntax

controlname.**InsertData** (*seriesnumber*, *datapoint*, *datavalue* {, *categoryvalue* })

Argument	Description
controlname	The name of the graph in which you want to insert data into a series.
seriesnumber	The number that identifies the series in which you want to insert data.
datapoint	The number of the data point before which you want to insert the data.
datavalue	The value of the data point you want to insert.
categoryvalue (optional)	The category for this data value on the category axis. The datatype of categoryvalue should match the datatype of the category axis. In most cases, you should include *categoryvalue*. Otherwise, an uncategorized value will be added to the series.

Return value

Integer. Returns the number of the data value if it succeeds and -1 if an error occurs. If any argument's value is null, InsertData returns null.

Usage

When you specify *datapoint* without specifying *categoryvalue*, InsertData inserts the data point in the category at that position, shifting existing data points to the following categories. The shift may cause there to be uncategorized data points at the end of the axis.

When you specify *categoryvalue*, InsertData ignores the position in *datapoint* and puts the data point in the specified category, replacing any data value that is already there. If the category does not exist, InsertData creates the category at the end of the axis.

To modify the value of a data point at a specified position, use ModifyData.

Scatter graphs
To add data to a scatter graph, use Syntax 2 of AddData.

Equivalent syntax If you want to add a data point to the end of a series or to an existing category in a series, you can use AddData instead, which requires fewer arguments.

InsertData and ModifyData behave differently when you specify *datapoint* to indicate a position for inserting or modifying data. However, they behave the same as AddData when you specify a position of 0 and a category. All three modify the value of a data point when the category already exists. All three insert a category with a data value at the end of the axis when the category does not exist.

When you specify a position as well as a category, and that category already exists, InsertData ignores the position and modifies the data of the specified category, but ModifyData changes the category label at that position.

This statement:

```
gr_data.InsertData(1, 0, 44, "Qty")
```

is equivalent to:

```
gr_data.ModifyData(1, 0, 44, "Qty")
```

and is also equivalent to:

```
gr_data.AddData(1, 44, "Qty")
```

When you specify a position, the following statements are not equivalent:

- InsertData ignores the position and modifies the data value of the Qty category:

  ```
  gr_data.InsertData(1, 4, 44, "Qty")
  ```

- ModifyData changes the category label and the data value at position 4:

  ```
  gr_data.ModifyData(1, 4, 44, "Qty")
  ```

Examples

Assuming the category label Jan does not already exist, these statements insert a data value in the series named Costs before the data point for Mar and assign the data point the category label Jan in the graph gr_product_data:

```
integer SeriesNbr, CategoryNbr

// Get the numbers of the series and category.
SeriesNbr = gr_product_data.FindSeries("Costs")
CategoryNbr = gr_product_data.FindCategory("Mar")
```

```
gr_product_data.InsertData(SeriesNbr, &
    CategoryNbr, 1250, "Jan")
```

These statements insert the data value 1250 after the data value for Apr in the series named Revenues in the graph gr_product_data. The data is inserted in the category after Apr, and the rest of the data, if any, moves over a category:

```
integer SeriesNbr, CategoryNbr

// Get the number of the series and category.
CategoryNbr = gr_product_data.FindCategory("Apr")
SeriesNbr = gr_product_data.FindSeries("Revenues")

gr_product_data.InsertData(SeriesNbr, &
    CategoryNbr + 1, 1250)
```

See also

AddData
FindCategory
FindSeries
GetData

InsertDocument

Description

Inserts a rich text format or plain text file into a RichTextEdit control, DataWindow control, or DataStore object.

PocketBuilder	✗
PowerBuilder	✓

Applies to

RichTextEdit controls, DataWindow controls, and DataStore objects

Syntax

rtename.**InsertDocument** (*filename*, *clearflag* { , *filetype* })

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is null, InsertDocument returns null.

InsertFile

Description

Inserts an object into an OLE control. A copy of the specified file is embedded in the OLE object.

PocketBuilder	✗
PowerBuilder	✓

Syntax

olecontrol.**InsertFile** (*filename*)

Return value

Integer. Returns 0 if it succeeds and a negative number if an error occurs.

InsertItem

Inserts an item into a ListBox, DropDownListBox, ListView, TreeView or Toolbar control.

To insert an item into a	Use
ListBox or DropDownListBox control	Syntax 1
PictureListBox or DropDownPictureListBox control	Syntax 2
ListView control when only the label and picture index need to be specified	Syntax 3
ListView control when more than the label and picture index need to be specified	Syntax 4
TreeView control when only the label and picture index need to be specified	Syntax 5
TreeView control when more than the label and picture index need to be specified	Syntax 6
Toolbar control	Syntax 7

Syntax 1 For ListBox and DropDownListBox controls

Description

Inserts an item into the list of values in a list box.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

ListBox and DropDownListBox controls

Syntax

listboxname.**InsertItem** (*item*, *index*)

Argument	Description
listboxname	The name of the ListBox or DropDownListBox into which you want to insert an item
item	A string whose value is the text of the item you want to insert
index	The number of the item in the list before which you want to insert the item

Return value

Integer. Returns the final position of the item. Returns -1 if an error occurs. If any argument's value is null, InsertItem returns null.

Usage

InsertItem inserts the new item before the item identified by *index*. If the items in *listboxname* are sorted (its Sorted property is true), PocketBuilder resorts the items after the new item is inserted. The return value reflects the new item's final position in the list.

AddItem and InsertItem do not update the Items property array. You can use FindItem to find items added during execution.

Examples

This statement inserts the item Run Application before the fifth item in lb_actions:

```
lb_actions.InsertItem("Run Application", 5)
```

If the Sorted property is false, the statement above returns 5 (the previous item 5 becomes item 6). If the Sorted property is true, the list is sorted after the item is inserted and the function returns the index of the final position of the item.

If the ListBox lb_Cities has the following items in its list and its Sorted property is set to true, then the following example inserts Denver at the top, sorts the list, and sets *li_pos* to 4. If the ListBox's Sorted property is false, then the statement inserts Denver at the top of the list and sets *li_pos* to 1. The list is:

```
Albany
Boston
Chicago
New York
```

The example code is:

```
string ls_City = "Denver"
integer li_pos
li_pos = lb_Cities.InsertItem(ls_City, 1)
```

See also

AddItem
DeleteItem
FindItem
Reset
TotalItems

Syntax 2 For ListBox and DropDownListBox controls

Description

Inserts an item into the list of values in a picture list box.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

PictureListBox and DropDownPictureListBox controls

Syntax

listboxname.**InsertItem** (*item* {, *pictureindex* }, *index*)

Argument	Description
listboxname	The name of the PictureListBox or DropDownPictureListBox into which you want to insert an item
item	A string whose value is the text of the item you want to insert
pictureindex (optional)	An integer specifying the index of the picture you want to associate with the newly added item
index	The number of the item in the list before which you want to insert the item

Return value

Integer. Returns the final position of the item. Returns -1 if an error occurs. If any argument's value is null, InsertItem returns null.

Usage

If you do not specify a picture index, the newly added item will not have a picture.

If you specify a picture index that does not exist, that number is still stored with the picture. If you add pictures to the picture array so that the index becomes valid, the item will then show the corresponding picture.

For additional notes about items in ListBoxes and examples of how the Sorted property affects the item order, see Syntax 1.

Examples

This statement inserts the item Run Application before the fifth item in lb_actions. The item has no picture assigned to it:

```
plb_actions.InsertItem("Run Application", 5)
```

This statement inserts the item Run Application before the fifth item in lb_actions and assigns it picture index 4:

```
plb_actions.InsertItem("Run Application", 4, 5)
```

See also

AddItem
DeleteItem
FindItem
Reset
TotalItems

Syntax 3 For ListView controls

Description

Inserts an item into a ListView control.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

ListView controls

Syntax

listviewname.**InsertItem** (*index, label, pictureindex*)

Argument	Description
listviewname	The name of the ListView control to which you are adding an item
index	An integer whose value is the index number of the item before which you are inserting a new item
label	A string whose value is the name of the item you are adding
pictureindex	An integer whose value is the index number of the picture of the item you are adding

Return value

Integer. Returns *index* if it succeeds and -1 if an error occurs.

Usage

If you need to set more than the label and picture index, use Syntax 4.

Examples

This example inserts an item in the ListView in position 11:

```
lv_list.InsertItem(11 , "Presentation" , 1)
```

See also

AddItem

Syntax 4 For ListView controls

Description

Inserts an item into a ListView control.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

ListView controls

Syntax

listviewname.**InsertItem** (*index, item*)

Argument	Description
listviewname	The name of the ListView control into which you are inserting an item
index	An integer whose value is the index number of the item you are adding
item	A system structure of datatype ListViewItem in which InsertItem stores the item you are inserting

Return value

Integer. Returns *index* if it succeeds and -1 if an error occurs.

Usage

The index you specify is the position of the item you are adding to a ListView.

If you need to insert just the label and picture index into the ListView control, use Syntax 3.

Examples

This example moves a ListView item from the second position into the fifth position. It uses GetItem to retrieve the state information from item 2, inserts it into the ListView control as item 5, and then deletes the original item:

```
listviewitem l_lvi

lv_list.GetItem(2 , l_lvi)
lv_list.InsertItem(5 , l_lvi)
lv_list.DeleteItem(2)
```

See also

AddItem

Syntax 5 For TreeView controls

Description

Inserts an item at a specific level and order in a TreeView control.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

TreeView controls

Syntax

treeviewname.**InsertItem** (*handleparent, handleafter, label, pictureindex*)

Argument	Description
treeviewname	The name of the TreeView control in which you want to insert an item.
handleparent	The handle of the item one level above the item you want to insert. To insert an item at the first level, specify 0.
handleafter	The handle of the item on the same level that you will insert the item immediately after.
label	The label of the item you are inserting.
pictureindex	The Index of the index of the picture you are adding to the image list.

Return value

Long. Returns the handle of the inserted item if it succeeds and -1 if an error occurs.

Usage

Use this syntax to set just the label and picture index. Use the next syntax if you need to set additional properties for the item.

If the TreeView's SortType property is set to a value other than Unsorted!, the inserted item is sorted with its siblings.

If you are inserting the first child of an item, use InsertItemLast or InsertItemFirst instead. Those functions do not require a *handleafter* value.

Examples

This example inserts a TreeView item that is on the same level as the current TreeView item. It uses FindItem to get the current item and its parent, then inserts the new item beneath the parent item:

```
long ll_tvi, ll_tvparent
ll_tvi = tv_list.FindItem(currenttreeitem! , 0)
ll_tvparent = tv_list.FindItem(parenttreeitem!,ll_tvi)
tv_list.InsertItem(ll_tvparent,ll_tvi,"Hindemith", 2)
```

See also

GetItem

Syntax 6 For TreeView controls

Description

Inserts an item at a specific level and order in a TreeView control.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

TreeView controls

Syntax

treeviewname.**InsertItem** (*handleparent, handleafter, item*)

Argument	Description
treeviewname	The name of the TreeView control into which you want to insert an item.
handleparent	The handle of the item one level above the item you want to insert. To insert an item at the first level, specify 0.
handleafter	The handle of the item on the same level that you will insert the item immediately after.
item	A TreeViewItem structure for the item you are inserting.

Return value

Long. Returns the handle of the item inserted if it succeeds and -1 if an error occurs.

Usage

Use the previous syntax to set just the label and picture index. Use this syntax if you need to set additional properties for the item.

If the TreeView's SortType property is set to a value other than Unsorted!, the inserted item is sorted with its siblings.

If you are inserting the first child of an item, use InsertItemLast or InsertItemFirst instead. Those functions do not require a *handleafter* value.

Examples

This example inserts a TreeView item that is on the same level as the current TreeView item. It uses FindItem to get the current item and its parent, then inserts the new item beneath the parent item:

```
long ll_tvi, ll_tvparent
treeviewitem  l_tvi

ll_tvi = tv_list.FindItem(currenttreeitem! , 0)
ll_tvparent = tv_list.FindItem(parenttreeitem!,ll_tvi)
tv_list.GetItem(ll_tvi , l_tvi)
tv_list.InsertItem(ll_tvparent,ll_tvi, l_tvi)
```

See also

GetItem

Syntax 7 For Toolbar controls

Description

Inserts a toolbar item in a toolbar control.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✗

Applies to

Toolbar controls

Syntax

Long *controlname*.InsertItem (*item*, *index*)

Argument	Description
controlname	The name of the toolbar control
item	Object of type ToolbarItem that you want to insert in a toolbar
index	Integer for the position where you want to insert the item in the toolbar

Return value

Long. Returns 1 for success and -1 if an error occurs.

Examples

The following example inserts an item at the third position in a toolbar:

```
Long ll_rtn
ToolbarItem myItem
myItem.ItemPictureIndex = 5
myItem.ItemStyle = stylecheck!
ll_rtn=tlbr_mytoolbar.InsertItem(myItem, 3)
```

See also

AddItem
DeleteItem
GetItem

InsertItemFirst

Inserts an item as the first child of a parent item.

To insert an item as the first child of its parent	Use
When you only need to specify the item label and picture index	Syntax 1
When you need to specify more than the item label and picture index	Syntax 2

Syntax 1 For TreeView controls

Description

Inserts an item as the first child of its parent.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

TreeView controls

Syntax

treeviewname.**InsertItemFirst** (*handleparent, label, pictureindex*)

Argument	Description
treeviewname	The TreeView control in which you want to specify an item as the first child of its parent.
handleparent	The handle of the item that will be the inserted item's parent. To insert the item at the first level, specify 0.
label	The label of the item you want to specify as the first child of its parent.
pictureindex	The picture index for the item you want to specify as the first child of its parent.

Return value

Long. Returns the handle of the item inserted if it succeeds and -1 if an error occurs.

Examples

This example populates the first level of a TreeView using InsertItemFirst:

```
long ll_lev1, ll_lev2 ,ll_lev3 ,ll_lev4
int index

tv_list.PictureHeight = 32
tv_list.PictureWidth = 32

ll_lev1 = tv_list.InsertItemFirst(0,"Composers",1)
ll_lev2 = tv_list.InsertItemLast(ll_lev1, &
    "Beethoven",2)
ll_lev3 = tv_list.InsertItemLast(ll_lev2, &
    "Symphonies", 3)

FOR index = 1 to 9
    ll_lev4 = tv_list.InsertItemSort(ll_lev3, &
      "Symphony # " + String(index) , 4)
NEXT

tv_list.ExpandItem(ll_lev3)
tv_list.ExpandItem(ll_lev4)
```

See also

InsertItem
InsertItemLast
InsertItemSort

Syntax 2 For TreeView controls

Description

Inserts an item as the first child of an item.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

TreeView controls

Syntax

treeviewname.**InsertItemFirst** (*handleparent, item*)

Argument	Description
treeviewname	The TreeView control in which you want to specify an item as the first child of its parent.
handleparent	The handle of the item that will be the inserted item's parent. To insert the item at the first level, specify 0.
item	A TreeViewItem structure for the item you are inserting.

Return value

Long. Returns the handle of the item inserted if it succeeds and -1 if an error occurs.

Usage

If SortType is anything except Unsorted!, items are sorted after they are added and the TreeView is always in a sorted state. Therefore, calling InsertItemFirst, InsertItemLast, and InsertItemSort produces the same result.

Examples

This example inserts the current item as the first item beneath the root item in a TreeView control:

```
long          ll_handle, ll_roothandle
treeviewitem  l_tvi
ll_handle = tv_list.FindItem(CurrentTreeItem!, 0)
ll_roothandle = tv_list.FindItem(RootTreeItem!, 0)
tv_list.GetItem(ll_handle , l_tvi)

tv_list.InsertItemFirst(ll_roothandle, l_tvi)
```

See also

InsertItem
InsertItemLast
InsertItemSort

InsertItemLast

Inserts an item as the last child of a parent item.

To insert an item as the last child of its parent	Use
When you only need to specify the item label and picture index	Syntax 1
When you need to specify more than item label and picture index	Syntax 2

Syntax 1 For TreeView controls

Description

Inserts an item as the last child of its parent.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

TreeView controls

Syntax

treeviewname.**InsertItemLast** (*handleparent, label, pictureindex*)

Argument	Description
treeviewname	The TreeView control in which you want to specify an item as the last child of its parent.
handleparent	The handle of the item that will be the inserted item's parent. To insert the item at the first level, specify 0.
label	The label of the item you want to specify as the last child of its parent.
pictureindex	The picture index for the item you want to specify as the last child of its parent.

Return value

Long. Returns the handle of the item inserted if it succeeds and -1 if an error occurs.

Usage

If more than the item label and Index need to be specified, use syntax 2.

If SortType is anything except Unsorted!, items are sorted after they are added and the TreeView is always in a sorted state. Therefore, calling InsertItemFirst, InsertItemLast, and InsertItemSort produces the same result.

Examples

This example populates the first three levels of a TreeView using InsertItemLast:

```
long  ll_lev1, ll_lev2, ll_lev3, ll_lev4
```

```
int   index

tv_list.PictureHeight = 32
tv_list.PictureWidth = 32

ll_lev1 = tv_list.InsertItemLast(0,"Composers",1)
ll_lev2 = tv_list.InsertItemLast(ll_lev1, &
    "Beethoven",2)
ll_lev3 = tv_list.InsertItemLast(ll_lev2, &
    "Symphonies",3)
FOR index = 1 to 9
    ll_lev4 = tv_list.InsertItemSort(ll_lev3, &
      "Symphony # " String(index), 4)
NEXT

tv_list.ExpandItem(ll_lev3)
tv_list.ExpandItem(ll_lev4)
```

See also

InsertItem
InsertItemFirst
InsertItemSort

Syntax 2 For TreeView controls

Description

Inserts an item as the last child of its parent.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

TreeView controls

Syntax

treeviewname.**InsertItemLast** (*handleparent, item*)

Argument	Description
treeviewname	The TreeView control in which you want to specify an item as the last child of its parent.
handleparent	The handle of the item that will be the inserted item's parent. To insert the item at the first level, specify 0.
item	A TreeViewItem structure for the item you are inserting.

Return value

Long. Returns the handle of the item inserted if it succeeds and -1 if an error occurs.

Usage

If SortType is anything except Unsorted!, items are sorted after they are added and the TreeView is always in a sorted state. Therefore, calling InsertItemFirst, InsertItemLast, and InsertItemSort produces the same result.

Examples

This example inserts the current item as the last item beneath the root item in a TreeView control:

```
long         ll_handle, ll_roothandle
treeviewitem l_tvi

ll_handle = tv_list.FindItem(CurrentTreeItem!, 0)
ll_roothandle = tv_list.FindItem(RootTreeItem!, 0)
tv_list.GetItem(ll_handle , l_tvi)

tv_list.InsertItemLast(ll_roothandle, l_tvi)
```

See also

InsertItem
InsertItemFirst
InsertItemSort

InsertItemSort

Inserts a child item in sorted order under the parent item.

To insert an item in sorted order	Use
When you only need to specify the item label and picture index	Syntax 1
When you need to specify more than the item label and picture index	Syntax 2

Syntax 1 For TreeView controls

Description

Inserts an item in sorted order, if possible.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

TreeView controls

Syntax

treeviewname.**InsertItemSort** (*handleparent, label, pictureindex*)

Argument	Description
treeviewname	The TreeView control in which you want to insert and sort an item as a child of its parent, according to its label.
handleparent	The handle of the item that will be the inserted item's parent. To insert the item at the first level, specify 0.
label	The label by which you want to sort the item as a child of its parent.
pictureindex	The picture index for the item you want to sort as a child of its parent, according to its label.

Return value

Long. Returns the handle of the item inserted if it succeeds and -1 if an error occurs.

Usage

If SortType is anything except Unsorted!, the TreeView is always in a sorted state and you do not need to use InsertItemSort—you can use any insert function.

If SortType is Unsorted!, InsertItemSort attempts to insert the item at the correct place in alphabetic ascending order. If the list is out of order, it does its best to find the correct place, but results may be unpredictable.

Examples

This example populates the fourth level of a TreeView control:

```
long ll_lev1, ll_lev2, ll_lev3, ll_lev4
int  index

tv_list.PictureHeight = 32
tv_list.PictureWidth = 32

ll_lev1 = tv_list.InsertItemLast(0,"Composers",1)
ll_lev2 = tv_list.InsertItemLast(ll_lev1,&
    "Beethoven",2)
ll_lev3 = tv_list.InsertItemLast(ll_lev2,&
    "Symphonies",3)
FOR index = 1 to 9
    ll_lev4 = tv_list.InsertItemSort(ll_lev3, &
      "Symphony # " + String(index), 4)
NEXT

tv_list.ExpandItem(ll_lev3)
tv_list.ExpandItem(ll_lev4)
```

See also

InsertItem
InsertItemLast
InsertItemFirst

Syntax 2 For TreeView controls

Description

Inserts an item in sorted order, if possible.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

TreeView controls

Syntax

treeviewname.**InsertItemSort** (*handleparent, item*)

Argument	Description
treeviewname	The TreeView control in which you want to sort an item as a child of its parent, according to its label.
handleparent	The handle of the item that will be the inserted item's parent. To insert the item at the first level, specify 0.
item	A TreeViewItem structure for the item you are inserting.

Return value

Long. Returns the handle of the item inserted if it succeeds and -1 if an error occurs.

Usage

If SortType is anything except Unsorted!, the TreeView is always in a sorted state and you do not need to use InsertItemSort—you can use any insert function.

If SortType is Unsorted!, InsertItemSort attempts to insert the item at the correct place in alphabetic ascending order. If the list is out of order, it does its best to find the correct place, but results may be unpredictable.

Examples

This example inserts the current item beneath the root item in a TreeView control and sorts it according to its label:

```
long ll_handle, ll_roothandle
treeviewitem l_tvi

ll_handle = tv_list.FindItem(CurrentTreeItem!, 0)
ll_roothandle = tv_list.FindItem(RootTreeItem!, 0)
tv_list.GetItem(ll_handle , l_tvi)

tv_list.InsertItemSort(ll_roothandle, l_tvi)
```

See also

InsertItem
InsertItemFirst
InsertItemLast

InsertObject

Description

Displays the standard Insert Object dialog box, allowing the user to choose a new or existing OLE object, and inserts the selected object in the OLE control.

PocketBuilder	✗
PowerBuilder	✓

Syntax

olecontrol.**InsertObject** ( )

Return value

Integer. Returns 0 if it succeeds and one of the following values if an error occurs:

1	User canceled out of dialog box
-9	Error

If any argument's value is null, InsertObject returns null.

InsertPicture

Description

Inserts a bitmap at the insertion point in a RichTextEdit control.

PocketBuilder	✗
PowerBuilder	✓

Applies to

RichTextEdit controls

Syntax

rtename.**InsertPicture** (*filename*)

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs. If *filename* is null, InsertPicture returns null.

InsertSeries

Description

Inserts a series in a graph at the specified position. Existing series in the graph are renumbered to keep the numbering sequential.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Graph controls in windows and user objects. Does not apply to graphs within DataWindow objects, because their data comes directly from the DataWindow.

Syntax

controlname.**InsertSeries** (*seriesname*, *seriesnumber*)

Argument	Description
controlname	The name of the graph in which you want to insert a series.
seriesname	A string containing the name of the series you want to insert. The series name must be unique within the graph.
seriesnumber	The number of the series before which you want to insert the new series. To add the new series at the end, specify 0.

Return value

Integer. Returns the number of the series if it succeeds and -1 if an error occurs. If the series named in *seriesname* exists already, it returns the number of the existing series. If any argument's value is null, InsertSeries returns null.

Usage

Series names are unique if they have different capitalization.

Equivalent syntax If you want to add a series to the end of the list, you can use AddSeries instead, which requires fewer arguments.

This statement:

```
gr_data.InsertSeries("Costs", 0)
```

is equivalent to:

```
gr_data.AddSeries("Costs")
```

Examples

These statements insert a series before the series named Income in the graph gr_product_data:

```
integer SeriesNbr

// Get the number of the series.
SeriesNbr = FindSeries("Income")
gr_product_data.InsertSeries("Costs", SeriesNbr)
```

See also

AddData
AddSeries
FindCategory
FindSeries
InsertCategory
InsertData

Int

Description

Determines the largest whole number less than or equal to a number.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

Int (*n*)

Argument	Description
n	The number for which you want the largest whole number that is less than or equal to it

Return value

Integer. Returns the largest whole number less than or equal to *n*. If *n* is too small or too large to be represented as an integer, Int returns 0. If *n* is null, Int returns null.

Usage

When the result for Int would be smaller than -32768 or larger than 32767, Int returns 0 because the result cannot be represented as an integer.

Examples

These statements return 3.0:

```
Int(3.2)
Int(3.8)
```

The following statements return -4.0:

```
Int(-3.2)
Int(-3.8)
```

These statements remove the decimal portion of the variable and store the resulting integer in *li_nbr*:

```
integer li_nbr
li_nbr = Int(3.2) // li_nbr = 3
```

See also

Ceiling
Round
Truncate
Int method for DataWindows in the *DataWindow Reference* or the online Help

Integer

Description

Converts the value of a string to an integer or obtains an integer value that is stored in a blob.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

Integer (*stringorblob*)

Argument	Description
stringorblob	A string whose value you want returned as an integer or a blob in which the first value is the integer value. The rest of the contents of the blob is ignored. *Stringorblob* can also be an Any variable containing a string or blob.

Return value

Integer. Returns the value of *stringorblob* as an integer if it succeeds and 0 if *stringorblob* is not a valid number or is an incompatible datatype. If *stringorblob* is null, Integer returns null.

Usage

To distinguish between a string whose value is the number 0 and a string whose value is not a number, use the IsNumber function before calling the Integer function.

Examples

This statement returns the string 24 as an integer:

```
Integer("24")
```

This statement returns the contents of the SingleLineEdit sle_Age as an integer:

```
Integer(sle_Age.Text)
```

This statement returns 0:

```
Integer("3ABC") // 3ABC is not a number.
```

This statement (PocketBuilder only) returns a decimal value for a hexadecimal string :

```
Integer("0x3ABC") // Returns 15036.
```

This example checks whether the text of sle_data is a number before converting, which is necessary if the user might legitimately enter 0:

```
integer li_new_data
IF IsNumber(sle_data.Text) THEN
    li_new_data = Integer(sle_data.Text)
```

```
ELSE
    SetNull(li_new_data)
END IF
```

After assigning blob data from the database to lb_blob, this example obtains the integer value stored at position 20 in the blob:

```
integer i
i = Integer(BlobMid(lb_blob, 20, 2))
```

See also

Double
Dec
IsNumber
Long
Real
Integer method for DataWindows in the *DataWindow Reference*

InternetData

Description

Processes the HTML data returned by a GetURL or PostURL function. The Context object calls this function; you do not call this function explicitly. Instead, you override this function in a customized descendant of the InternetResult standard class user object.

PocketBuilder	✗
PowerBuilder	✓

Applies to

InternetResult objects

Syntax

servicereference.**InternetData** (*data*)

Return value

Integer. Returns 1 if the function succeeds and -1 if an error occurs.

IntHigh

Description

Returns the high word of a long value.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

IntHigh (*long*)

Argument	Description
long	A long value

Return value

Integer. Returns the high word of *long* if it succeeds and -1 if an error occurs. If *long* is null, IntHigh returns null.

Usage

One use for IntHigh is for decoding values returned by external C functions and Windows messages.

Examples

These statements decode a long value *LValue* into its low and high integers:

```
integer nLow, nHigh
long LValue = 274489
nLow = IntLow (LValue)  //The Low Integer is 12345.
nHigh = IntHigh(LValue) //The High Integer is 4.
```

See also

IntLow

IntLow

Description

Returns the low word of a long value.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

IntLow (*long*)

Argument	Description
long	A long value

Return value

Integer. Returns the low word of *long* if it succeeds and -1 if an error occurs. If *long* is null, IntLow returns null.

Usage

One use for IntLow is for decoding values returned by external C functions and Windows messages.

Examples

These statements decode a long value *LValue* into its low and high integers:

```
integer nLow, nHigh
long LValue = 12345
nLow = IntLow(LValue)   //The Low Integer is 12345.
nHigh = IntHigh(LValue) //The High Integer is 0.
```

See also	IntHigh

InvokePBFunction

Description	Invokes the specified user-defined window function in the child window contained in a PowerBuilder window ActiveX control.

PocketBuilder	✗
PowerBuilder	✓

Applies to	Window ActiveX controls
Syntax	*activexcontrol*.**InvokePBFunction** (*name* {, *numarguments* {, *arguments* } })
Return value	Integer. Returns 1 if the function succeeds and -1 if an error occurs.

_Is_A

Description	Checks to see whether a CORBA object is an instance of a class that implements a particular interface. This function is used by PowerBuilder clients connecting to EAServer.

PocketBuilder	✗
PowerBuilder	✓

Applies to	CORBAObject objects
Syntax	*corbaobject*.**_Is_A** (*classname*)
Return value	Boolean. Returns true if the class of the object implements the specified interface and false if it does not.

IsAlive

Description

Determines whether a server object is still running.

PocketBuilder	✗
PowerBuilder	✓

Applies to

OLEObject objects, OLETxnObject objects

Syntax

oleobject.**IsAlive** ( )

Return value

Boolean. Returns true if the server object appears to be running and false if it is dead.

IsAllArabic

Description

Tests whether a particular string is composed entirely of Arabic characters.

PocketBuilder	✗
PowerBuilder	✓

Syntax

IsAllArabic (*string*)

Return value

Boolean. Returns true if *string* is composed entirely of Arabic characters and false if it is not. The presence of numbers, spaces, and punctuation marks will also result in a return value of false.

IsAllHebrew

Description

Tests whether a particular string is composed entirely of Hebrew characters.

PocketBuilder	✗
PowerBuilder	✓

Syntax

IsAllHebrew (*string*)

Return value

Boolean. Returns true if *string* is composed entirely of Hebrew characters and false if it is not. The presence of numbers, spaces, and punctuation marks will also result in a return value of false.

IsAnyArabic

Description
Tests whether a particular string contains at least one Arabic character.

PocketBuilder	✗
PowerBuilder	✓

Syntax
IsAnyArabic (*string*)

Return value
Boolean. Returns true if *string* contains at least one Arabic character and false if it does not.

IsAnyHebrew

Description
Tests whether a particular string contains at least one Hebrew character.

PocketBuilder	✗
PowerBuilder	✓

Syntax
IsAnyHebrew (*string*)

Return value
Boolean. Returns true if *string* contains at least one Hebrew character and false if it does not.

IsArabic

Description
Tests whether a particular character is an Arabic character. For a string, IsArabic tests only the first character on the left.

PocketBuilder	✗
PowerBuilder	✓

Syntax
IsArabic (*character*)

Return value
Boolean. Returns true if *character* is an Arabic character and false if it is not.

IsArabicAndNumbers

Description Tests whether a particular string is composed entirely of Arabic characters or numbers.

PocketBuilder	✗
PowerBuilder	✓

Syntax **IsArabicAndNumbers** (*string*)

Return value Boolean. Returns true if *string* is composed entirely of Arabic characters or numbers and false if it is not.

IsCallerInRole

Description Indicates whether the direct caller of a COM object running on MTS is in a specified role (either individually or as part of a group).

PocketBuilder	✗
PowerBuilder	✓

Applies to TransactionServer objects

Syntax *transactionserver*.**IsCallerInRole** (*role*)

Return value Boolean. Returns true if the direct caller is in the specified role and false if it is not.

IsDate

Description Tests whether a string value is a valid date.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

IsDate (*datevalue*)

Argument	Description
datevalue	A string whose value you want to test to determine whether it is a valid date

Return value

Boolean. Returns true if *datevalue* is a valid date and false if it is not. If *datevalue* is null, IsDate returns null.

Usage

You can use IsDate to test whether a user-entered date is valid before you convert it to a date datatype. To convert a value into a date value, use the Date function.

Examples

This statement returns true:

```
IsDate("Jan 1, 95")
```

This statement returns false:

```
IsDate("Jan 32, 1997")
```

If the SingleLineEdit sle_Date_Of_Hire contains 7/1/91, these statements store 1991-07-01 in *HireDate*:

```
Date HireDate
IF IsDate(sle_Date_Of_Hire.text) THEN
    HireDate = Date(sle_Date_Of_Hire.text)
END IF
```

See also

IsDate method for DataWindows in the *DataWindow Reference* or the online Help

IsHebrew

Description

Tests whether a particular character is a Hebrew character. For a string, IsHebrew tests only the first character on the left.

PocketBuilder	✗
PowerBuilder	✓

Syntax

IsHebrew (*character*)

Return value

Boolean. Returns true if *character* is an Hebrew character and false if it is not.

IsHebrewAndNumbers

Description — Tests whether a particular string is composed entirely of Hebrew characters and numbers.

PocketBuilder	✗
PowerBuilder	✓

Syntax — **IsHebrewAndNumbers** (*string*)

Return value — Boolean. Returns true if *string* is composed entirely of Hebrew characters and numbers and false if it is not.

IsImpersonating

Description — Queries whether a COM object running on MTS is impersonating the client.

PocketBuilder	✗
PowerBuilder	✓

Applies to — TransactionServer objects

Syntax — *transactionserver*.**IsImpersonating** ( )

Return value — Boolean. Returns true if the component is impersonating the client and false if it is not.

IsInTransaction

Description — Indicates whether a component is executing in a transaction.

PocketBuilder	✗
PowerBuilder	✓

Applies to — TransactionServer objects

Syntax — *transactionserver*.**IsInTransaction** ( )

Return value — Boolean. Returns true if the component is executing as part of a transaction and false if it is not.

IsNull

Description

Reports whether the value of a variable or expression is null.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

IsNull (*any*)

Argument	Description
any	A variable or expression that you want to test to determine whether its value is null

Return value

Boolean. Returns true if *any* is null and false if it is not.

Usage

Use IsNull to test whether a user-entered value or a value retrieved from the database is null. IsNull works for all datatypes but does not work for arrays.

If one or more columns in a DataWindow are required columns, that is, they must contain data, you do not want to update the database if the columns have null values. You can use FindRequired to find rows in which those columns have null values, instead of using IsNull to evaluate each row and column.

Setting a variable to null
To set a variable to null, use the SetNull function.

Examples

These statements set *lb_test* to true:

```
integer a, b
boolean lb_test
SetNull(b)
lb_test = IsNull(a + b)
```

See also

SetNull
IsNull method for DataWindows in the *DataWindow Reference*

IsNumber

Description

Reports whether the value of a string is a number.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

IsNumber (*string*)

Argument	Description
string	A string whose value you want to test to determine whether it is a valid PowerScript number

Return value

Boolean. Returns true if *string* is a valid PowerScript number and false if it is not. If *string* is null, IsNumber returns null.

Usage

Use IsNumber to check that text in an edit control can be converted to a number.

To convert a string to a specific numeric datatype, use the Double, Dec, Integer, Long, or Real function.

Examples

This statement returns true:

```
IsNumber("32.65")
```

This statement returns false:

```
IsNumber("A16")
```

If the SingleLineEdit sle_Age contains 32, these statements store 32 in *li_YearsOld*:

```
integer li_YearsOld
IF IsNumber(sle_Age.Text) THEN
    li_YearsOld = Integer(sle_Age.Text)
END IF
```

See also

Double
Dec
Integer
Long
Real
IsNumber method for DataWindows in the *DataWindow Reference*

IsPreview

Description

Reports whether a RichTextEdit control is in preview mode.

PocketBuilder	✗
PowerBuilder	✓

Applies to

RichTextEdit controls

Syntax

rtename.**IsPreview** ( )

Return value

Boolean. Returns true if *rtename* is in preview mode and false if it is in data entry mode.

IsSecurityEnabled

Description

Indicates whether or not security checking is enabled for a COM object running on MTS or COM+.

PocketBuilder	✗
PowerBuilder	✓

Applies to

TransactionServer objects

Syntax

transactionserver.**IsSecurityEnabled** ( )

Return value

Boolean. Returns true if security checking is enabled and false if it is not.

IsSIPVisible

Description

Indicates whether the SIP is currently visible to the user.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✗

Syntax

Boolean IsSIPVisible ( )

Return value

Boolean. Returns “true” if the SIP is currently visible and “false” if it is not currently visible.

Usage

You can use this method to report the status of the SIP window.

Examples

The following example displays the status of the SIP window in a single-line-edit text box:

```
IF isSIPVisible() THEN
    sle_test.text = "SIP is UP"
ELSE
    sle_test.text = "SIP is Down"
END IF
```

See also

GetSIPRect
GetSIPType
SetSIPPreferredState

IsTime

Description

Reports whether the value of a string is a valid time value.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

IsTime (*timevalue*)

Argument	Description
timevalue	A string whose value you want to test to determine whether it is a valid time

Return value

Boolean. Returns true if *timevalue* is a valid time and false if it is not. If *timevalue* is null, IsTime returns null.

Usage

Use IsTime to test to whether a value a user enters in an edit control is a valid time.

To convert a string to an time value, use the Time function.

Examples

This statement returns true:

```
IsTime("8:00:00 am")
```

This statement returns false:

```
IsTime("25:00")
```

If the SingleLineEdit sle_EndTime contains 4:15 these statements store 04:15:00 in *lt_QuitTime*:

```
Time lt_QuitTime
IF **IsTime** sle_EndTime.Text) THEN
    lt_QuitTime = Time(sle_EndTime.Text)
END IF
```

See also

Time
IsTime method for DataWindows in the DataWindow Reference or the online Help

IsTransactionAborted

Description

Determines whether the current transaction, in which an EAServer component participates, has been aborted.

PocketBuilder	✗
PowerBuilder	✓

Applies to

TransactionServer objects

Syntax

transactionserver.**IsTransactionAborted** ( )

Return value

Boolean. Returns true if the current transaction has been aborted and false if it has not.

IsValid

Description

Determines whether an object variable is instantiated—whether its value is a valid object handle.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

IsValid (*objectvariable*)

Argument	Description
objectvariable	An object variable or a variable of type Any—typically a reference to an object that you are testing for validity

Return value

Boolean. Returns true if *objectvariable* is an instantiated object. Returns false if *objectvariable* is not an object, or if it is an object that is not instantiated. If *objectvariable* is null, IsValid returns null.

Usage

Use IsValid instead of the Handle function to determine whether a window is open.

Examples

This statement determines whether the window w_emp is open and if it is not, opens it:

```
IF IsValid(w_emp) = FALSE THEN Open(w_emp)
```

This example returns -1 because the IsValid function returns false. Although the *objectvariable* argument is a valid string, it is not an instantiated object. The IsValid method would return true only if *la_value* was an instantiated object:

```
any la_value

la_value = "I’m a string"
IF NOT IsValid(la_value) THEN return -1
```

See also

Handle

KeyDown

Description

Determines whether the user pressed the specified key on the computer keyboard.

PocketBuilder	✗
PowerBuilder	✓

Syntax

KeyDown (*keycode*)

Return value

Boolean. Returns true if *keycode* was pressed and false if it was not. If *keycode* is null, KeyDown returns null.

LastPos

Description

Finds the last position of a target string in a source string.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

LastPos (*string1*, *string2* {, *searchlength* })

Argument	Description
string1	The string in which you want to find *string2*.
string2	The string you want to find in *string1*.
searchlength (optional)	A long that limits the search to the leftmost *searchlength* characters of the source string *string1*. The default is the entire string.

Return value

Long. Returns a long whose value is the starting position of the last occurrence of *string2* in *string1* within the characters specified in *searchlength*. If *string2* is not found in *string1* or if *searchlength* is 0, LastPos returns 0. If any argument's value is null, LastPos returns null.

Usage

The LastPos function is case sensitive. The entire target string must be found in the source string.

Examples

This statement returns 6, because the position of the last occurrence of RU is position 6:

```
LastPos("BABE RUTH", "RU")
```

This statement returns 3:

```
LastPos("BABE RUTH", "B")
```

This statement returns 0, because the case does not match:

```
LastPos("BABE RUTH", "be")
```

This statement searches the leftmost 4 characters and returns 0, because the only occurrence of RU is after position 4. The search length must be at least 7 (to include the complete string RU) before the statement returns 6 for the starting position of the last occurence of RU:

```
LastPos("BABE RUTH", "RU", 4)
```

These statements change the text in the SingleLineEdit sle_group. The last instance of the text NY is changed to North East:

```
long place_nbr
place_nbr = LastPos(sle_group.Text, "NY")
sle_group.SelectText(place_nbr, 2 )
sle_group.ReplaceText("North East")
```

These statements separate the return value of GetBandAtPointer into the band name and row number. The LastPos function finds the position of the (last) tab in the string and the Left and Mid functions extract the information to the left and right of the tab:

```
string s, ls_left, ls_right
integer li_tab

s = dw_groups.GetBandAtPointer()
li_tab = LastPos(s, "~t")

ls_left = Left(s, li_tab - 1)
ls_right = Mid(s, li_tab + 1)
```

These statements tokenize a source string backwards:

```
// Tokenize the source string backwards
// Results in "pbsyc90.dll  powerbuilder
// shared  sybase  programs  c:

string  sSource = &
  'c:\programs\sybase\shared\powerbuilder\pbsyc90.dll'
string  sFind   = '\'
string  sToken
long  llStart, llEnd

llEnd = Len(sSource) + 1
```

```
DO
   llStart = LastPos(sSource, sFind, llEnd)
   sToken = Mid(sSource, (llStart + 1), &
      (llEnd - llStart))
   mle_comment.text += sToken + '  '
   llEnd = llStart - 1
LOOP WHILE llStart > 1
```

See also

Pos

Left

Description

Obtains a specified number of characters from the beginning of a string.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

Left (*string*, *n*)

Argument	Description
string	The string you want to search
n	A long specifying the number of characters you want to return

Return value

String. Returns the leftmost *n* characters in *string* if it succeeds and the empty string ("") if an error occurs. If any argument's value is null, Left returns null. If *n* is greater than or equal to the length of the string, Left returns the entire string. It does not add spaces to make the return value's length equal to *n*.

Examples

This statement returns BABE:

```
Left("BABE RUTH", 4)
```

This statement returns BABE RUTH:

```
Left("BABE RUTH", 40)
```

These statements store the first 40 characters of the text in the SingleLineEdit sle_address in *emp_address*:

```
string emp_address
emp_address = Left(sle_address.Text, 40)
```

For sample code that uses Left to parse two tab-separated values, see the Pos function.

See also

Mid
Pos
Right
Left method for DataWindows in the *DataWindow Reference*

LeftW

Description

Obtains a specified number of characters from the beginning of a string.

PocketBuilder	✗
PowerBuilder	✓

Obsolete function
This function is obsolete. It has the same behavior as Left in all environments.

Syntax

LeftW (*string*, *n*)

Return value

String

LeftTrim

Description

Removes spaces from the beginning of a string.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

LeftTrim (*string*)

LeftTrimW (*string*)

Argument	Description
string	The string you want returned with leading spaces deleted

Return value

String. Returns a copy of *string* with leading spaces deleted if it succeeds and the empty string ("") if an error occurs. If *string* is null, LeftTrim returns null.

Examples

This statement returns RUTH:

```
LeftTrim(" RUTH")
```

These statements delete leading spaces from the text in the MultiLineEdit mle_name and store the result in *emp_name*:

```
string emp_name
emp_name = LeftTrim(mle_name.Text)
```

See also

RightTrim
Trim
LeftTrim method for DataWindows in the *DataWindow Reference*

LeftTrimW

Description

Removes spaces from the beginning of a string.

PocketBuilder	✗
PowerBuilder	✓

Obsolete function
This function is obsolete. It has the same behavior as LeftTrim in all environments.

Syntax

LeftTrimW (*string*)

Return value

String. Returns a copy of *string* with leading spaces deleted if it succeeds and the empty string ("") if an error occurs. If *string* is null, LeftTrimW returns null.

Len

Description

Reports the length of a string or a blob.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

Len (*stringorblob*)

Argument	Description
stringorblob	The string or blob for which you want the length in number of characters or in number of bytes

Return value

Long. Returns a long whose value is the length of *stringorblob* if it succeeds and -1 if an error occurs. If *stringorblob* is null, Len returns null.

Usage

Len counts the number of characters in a string. The null that terminates a string is not included in the count.

If you specify a size when you declare a blob, that is the size reported by Len. If you do not specify a size for the blob, Len initially reports the blob's length as 0. PocketBuilder assigns a size to the blob the first time you assign data to the blob. Len reports the length of the blob as the number characters it can contain.

Examples

This statement returns 0:

```
Len("")
```

These statements store in the variable *s_address_len* the length of the text in the SingleLineEdit sle_address:

```
long s_address_len
s_address_len = Len(sle_address.Text)
```

The following scenarios illustrate how the declaration of blobs affects their length, as reported by Len.

In the first example, an instance variable called *ib_blob* is declared but not initialized with a size. If you call Len before data is assigned to *ib_blob*, Len returns 0. After data is assigned, Len returns the blob's new length.

The declaration of the instance variable is:

```
blob ib_blob
```

The sample code is:

```
long ll_len
```

```
ll_len = Len(ib_blob)  // ll_len set to 0
ib_blob = Blob( "Test String")
ll_len = Len(ib_blob)   // ll_len set to 22
```

In the second example, *ib_blob* is initialized to the size 100 when it is declared. When you call Len for *ib_blob*, it always returns 100. This example uses BlobEdit, instead of Blob, to assign data to the blob because its size is already established. The declaration of the instance variable is:

```
blob{100} ib_blob
```

The sample code is:

```
long ll_len
ll_len = Len(ib_blob) // ll_len set to 100
BlobEdit(ib_blob, 1, "Test String")
ll_len = Len(ib_blob) // ll_len set to 100
```

See also

Len method for DataWindows in the *DataWindow Reference*

LenW

Description

Reports the length of a string or a blob.

PocketBuilder	✗
PowerBuilder	✓

Obsolete function
This function is obsolete. It has the same behavior as Len in all environments.

Syntax

LenW (*stringorblob*)

Length

Description

Reports the length in bytes of an open OLE stream.

PocketBuilder	✗
PowerBuilder	✓

Len function
To get the length of a string or blob, use the Len function.

Applies to	OLEStream objects
Syntax	*olestream*.**Length** (*sizevar*)
Return value	Integer. Returns 0 if it succeeds and a negative number if an error occurs.

LibraryCreate

Description

Creates an empty PocketBuilder or PowerBuilder library with optional comments.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

LibraryCreate (*libraryname* {, *comments* })

Argument	Description
libraryname	A string whose value is the name of the PocketBuilder library you want to create. If you want to create the library somewhere other than the current directory, enter the full path name.
comments (optional)	A string whose value is the comments you want to associate with the library.

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is null, LibraryCreate returns null.

Usage

LibraryCreate creates a PocketBuilder library file (PKL) in the current directory, unless you specify a directory path as part of *libraryname*. If you do not specify an extension, LibraryCreate adds the extension *.PKL*. In PowerBuilder, LibraryCreate creates a PowerBuilder library file (PBL).

Examples

This statement in a PowerBuilder application creates a library named dwTemp in the *PB* directory on drive C and associates a comment with the library:

```
LibraryCreate("c:\pb\dwTemp.pbl", &
   "Temporary library for dynamic DataWindows")
```

See also

LibraryDelete
LibraryDirectory
LibraryExport
LibraryImport

LibraryDelete

Description

Deletes a library file or, if you specify a DataWindow object, deletes the DataWindow object from the library.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

LibraryDelete (*libraryname* {, *objectname*, *objecttype* })

Argument	Description
libraryname	A string whose value is the name of the PocketBuilder library you want to delete or from which you want to delete a DataWindow object. If you do not specify a full path, LibraryDelete uses the system's standard file search order to find the file.
objectname (optional)	A string whose value is the name of the DataWindow object you want to delete from *libraryname*.
objecttype (optional)	A value of the LibImportType enumerated datatype identifying the type of object you want to delete. The only supported object type is ImportDataWindow!.

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is null, LibraryDelete returns null.

Usage

You can delete DataWindow objects from a library in a script with the LibraryDelete function. To delete other types of objects, use the Library painter.

Examples

This statement deletes a library called dwTemp in the current directory and on the current application library path:

```
LibraryDelete("dwTemp.pkl")
```

See also

LibraryCreate
LibraryDirectory
LibraryExport
LibraryImport

LibraryDirectory

Description

Obtains a list of the objects in a PowerBuilder library. The information provided is the object name, the date and time it was last modified, and any comments for the object. You can get a list of all objects or just objects of a specified type.

PocketBuilder	✗
PowerBuilder	✓

Syntax

LibraryDirectory (*libraryname*, *objecttype*)

Return value

String. LibraryDirectory returns a tab-separated list with one object per line. The format of the list is:

name ~t date/time modified ~t comments ~n

Returns the empty string ("") if an error occurs. If any argument's value is null, LibraryDirectory returns null.

LibraryDirectoryEx

Description

Obtains a list of the objects in a PowerBuilder library. The information provided is the object name, the date and time it was last modified, any comments for the object, and the object's type. You can get a list of all objects or just objects of a specified type.

PocketBuilder	✗
PowerBuilder	✓

Syntax

LibraryDirectoryEx (*libraryname*, *objecttype*)

Return value

String. LibraryDirectoryEx returns a tab-separated list with one object per line. The format of the list is:

name ~t date/time modified ~t comments ~t   type~n

Returns the empty string ("") if an error occurs. If any argument's value is null, LibraryDirectoryEx returns null.

LibraryExport

Description Exports an object from a library. The object is exported as syntax.

PocketBuilder	✗
PowerBuilder	✓

Syntax **LibraryExport** (*libraryname*, *objectname*, *objecttype*)

Return value String. Returns the syntax of the object if it succeeds. The syntax is the same as the syntax returned when you export an object in the Library painter except that LibraryExport does not include an export header. Returns the empty string ("") if an error occurs. If any argument's value is null, LibraryExport returns null.

LibraryImport

Description Imports a DataWindow object into a library. LibraryImport uses the syntax of the DataWindow object, which is specified in text format, to recreate the object in the library.

PocketBuilder	✗
PowerBuilder	✓

Syntax **LibraryImport** (*libraryname*, *objectname*, *objecttype*, *syntax*, *errors* {, *comments* })

Return value Integer. Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is null, LibraryImport returns null.

LineCount

Description Determines the number of lines in an edit control that allows multiple lines.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to RichTextEdit, MultiLineEdit, EditMask, and DataWindow controls

Syntax

editname.**LineCount** ( )

Argument	Description
editname	The name of the control for which you want the number of lines

Return value

Long. Returns the number of lines in *editname* if it succeeds and -1 if an error occurs. If *editname* is null, LineCount returns null.

Usage

LineCount counts each visible line, whether it was the result of wrapping or carriage returns.

When you call LineCount for a DataWindow, it reports the number of lines in the edit control over the current row and column. A user can enter multiple lines in a DataWindow column only if it has a text datatype and its box is large enough to display those lines. The size of the column's box determines the number of lines allowed in the column. When the user is typing, lines do not wrap automatically; the user must press enter to type additional lines.

In a MultiLineEdit control, lines wrap when the user's typing fills the control horizontally, unless either the HScrollBar or AutoHScroll property is true. If horizontal scrolling is enabled with these properties, the user must press enter to type additional lines.

A RichTextEdit control always contains an end-of-file mark even if there is no text in the control. Therefore, its line count is always at least 1. Other edit controls, when empty, have a line count of 0.

Examples

If the MultiLineEdit mle_Instructions has 9 lines, this example sets *li_Count* to 9:

```
integer li_Count
li_Count = mle_Instructions.LineCount()
```

These statements display a MessageBox if fewer than two lines have been entered in the MultiLineEdit mle_Address:

```
integer li_Lines
li_Lines = mle_Address.LineCount()
IF li_Lines < 2 THEN
    MessageBox("Warning", "2 lines are required.")
END IF
```

LineLength

Description

Determines the length of the line containing the insertion point in an edit control.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

RichTextEdit, MultiLineEdit, and EditMask controls

Syntax

editname.**LineLength** ( )

Argument	Description
editname	The name of the RichTextEdit, MultiLineEdit, or EditMask in which you want to determine the length of the line containing the insertion point

Return value

Long. Returns the length of the line containing the insertion point in *editname*. Returns -1 if an error occurs. If *editname* is null, LineLength returns null.

Usage

If the control contains a selection instead of a single insertion point, LineLength counts the line at the beginning of the selection.

PocketBuilder remembers where the insertion point is in each editable control. When the user moves the focus to another control, you can still find out the length of the line most recently edited by calling the LineLength function for that control.

Insertion point in editable controls
Because PocketBuilder remembers the position of the insertion point, users can resume editing at the insertion point if they make the control active by tabbing to it. When users make a control active by clicking on it, they move the insertion point as well.

For an EditMask control, LineLength reports the length of the mask, regardless of the number of characters the user has entered.

Examples

If the insertion point is positioned anywhere in line 5 of mle_Contact and line 5 contains the text Select All, *il_linelength* is set to 10 (the length of line 5):

```
integer li_linelength
li_linelength = mle_Contact.LineLength()
```

See also

Position
SelectedLine
SelectedStart
TextLine

LineList

Description

Provides a list of the lines in a routine included in a performance analysis model.

PocketBuilder on Desktop	✓
PocketBuilder on Pocket PC	✗
PocketBuilder on Smartphone	✗
PowerBuilder	✓

Applies to

ProfileRoutine object

Syntax

iinstancename.**LineList** (*list*)

Argument	Description
instancename	Instance name of the ProfileRoutine object.
list	An unbounded array variable of datatype ProfileLine in which LineList stores a ProfileLine object for each line in the routine. This argument is passed by reference.

Return value

ErrorReturn. Returns one of the following values:

- Success!—The function succeeded
- ModelNotExistsError!—The model does not exist

Usage

Use this function to extract a list of the lines in a routine included in the performance analysis model. You must have previously created the performance analysis model from a trace file using the BuildModel function. Each line is defined as a ProfileLine object and provides the number of times the line was hit, any calls made from the line, and the time spent on the line and in any called functions. The lines are listed in numeric order.

Lines are not returned for database statements and objects. If line information was not logged in the trace file, lines are not returned.

Examples

This example gets a list of the routines included in a performance analysis model and then gets a list of the lines in each routine:

```
Long ll_cnt
ProfileLine lproln_line[]

lpro_model.BuildModel()
lpro_model.RoutineList(iprort_list)

FOR ll_cnt = 1 TO UpperBound(iprort_list)
    iprort_list[ll_cnt].LineList(lproln_line)
    ...
NEXT
```

See also

BuildModel

LinkTo

Description

Establishes a link between an OLE control and a file or an item within the file.

PocketBuilder	✗
PowerBuilder	✓

Syntax

olecontrol.**LinkTo** (*filename* {, *sourceitem* })

Return value

Integer. Returns 0 if it succeeds and a negative number if an error occurs.

Log

Returns the natural logarithm of a number. For an ErrorLogging object, this function can be used to write a string to the log file maintained by the object's container.

To	Use
Determine the natural logarithm of a number	Syntax 1
Write a string to a log file	Syntax 2

Syntax 1 For all objects

Description

Determines the natural logarithm of a number.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

Log (*n*)

Argument	Description
n	The number for which you want the natural logarithm (base *e*). The value of *n* must be greater than 0.

Return value

Double. Returns the natural logarithm of *n*. An execution error occurs if *n* is negative or zero. If *n* is null, Log returns null.

Inverse of Log
The inverse of the Log function is the Exp function.

Examples

This statement returns 2.302585092:

```
Log(10)
```

This statement returns –.693147. . . :

```
Log(0.5)
```

Both these statements result in an error during execution:

```
Log(0)
Log(-2)
```

After the following statements execute, the value of *a* is 200:

```
double a, b =  Log(200)
a = Exp(b)// a = 200
```

See also

Exp
LogTen
Log method for DataWindows in the *DataWindow Reference*

Syntax 2 For ErrorLogging objects

Description

Writes a string to the log file maintained by the object's container.

PocketBuilder	✗
PowerBuilder	✓

Applies to

ErrorLogging objects

Syntax

errorlogobj.**Log** (*message*)

Return value

None.

Login

Description

Logs in to a POOM object, enabling a Pocket PC or Smartphone device user to perform operations relating to appointments, contacts, and tasks.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✗

Applies to

POOM objects

Syntax

Integer *objectname*.Login ({ *parentwindow* })

Argument	Description
objectname	The name of the POOM object
parentwindow (optional)	The name of the parent window or user object for the POOM object

Return value

Integer. Returns 1 for success and one of the following negative values if an error occurs:

-1 Unspecified error

-2 Cannot connect to the repository or a required internal subobject failed to connect to the repository

-3 Cannot log in to the repository

-4 Incorrect input argument

-5 Action cannot be performed

-6	The object identifier (OID) is not in the repository
-7	Feature is not implemented yet
-8	No matching entries found for the criteria

Usage

A user must be logged in to a POOM object to make any changes to or view any appointments, contacts, or tasks. The user must log out to remove the POOM object from memory.

Examples

The following example logs in to a POOM object:

```
Int li_rtn
POOM po_1
...
po_1 = CREATE POOM
li_rtn = po_1.login( )
```

See also

Logout

Logout

Description

Logs out of a POOM object, freeing the memory used by the object.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✗

Applies to

POOM objects

Syntax

objectname.Logout ( )

Argument	**Description**
objectname	The name of the POOM object

Return value

None.

Usage

The user must log out of a POOM object to remove it from memory.

Examples

The following example logs out of a POOM object:

```
po_1.logout( )
```

See also

Login

LogTen

Description

Determines the base 10 logarithm of a number.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

LogTen (*n*)

Argument	Description
n	The number for which you want the base 10 logarithm. The value of *n* must not be negative.

Usage

Double. Returns the base 10 logarithm of *n*. An execution error occurs if *n* is negative. If *n* is null, LogTen returns null.

Inverse of LogTen The expression `10^`***`n`*** is the inverse of `LogTen(`***`n`***`)`. To obtain the value of *n* in the equation ***`r`*** `= LogTen(`***`n`***`)`, use `n = 10^`***`r`***.

Examples

This statement returns 1:

```
LogTen(10)
```

The following statements both return 0:

```
LogTen(1)

LogTen(0)
```

This statement results in an execution error:

```
LogTen( – 2)
```

After the following statements execute, the value of a is 200:

```
double a, b = LogTen(200)
a = 10^b// a = 200
```

See also

Exp
LogTen
LogTen method for DataWindows in the *DataWindow Reference*

Long

Converts data into data of type long. There are two syntaxes.

To	Use
Combine two unsigned integers into a long value	Syntax 1
Convert a string whose value is a number into a long or to obtain a long value stored in a blob	Syntax 2

Syntax 1 For combining integers

Description

Combines two unsigned integers into a long value.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

Long (*lowword*, *highword*)

Argument	Description
lowword	An UnsignedInteger to be the low word in the long
highword	An UnsignedInteger to be the high word in the long

Return value

Long. Returns the long if it succeeds and -1 if an error occurs. If any argument's value is null, Long returns null.

Usage

Use Long for passing values to external C functions or specifying a value for the LongParm property of PocketBuilder's Message object.

Examples

These statements convert the UnsignedIntegers *nLow* and *nHigh* into a long value:

```
UnsignedInt nLow // Low integer 16 bits
UnsignedInt nHigh // High integer 16 bits
long LValue // Long value 32 bits

nLow = 12345
nHigh = 0
LValue =  Long(nLow, nHigh)
MessageBox("Long Value", Lvalue)
```

Syntax 2 For converting strings and blobs

Description

Converts a string whose value is a number into a long or obtains a long value stored in a blob.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

Long (*stringorblob*)

Argument	Description
stringorblob	The string you want returned as a long or a blob in which the first value is the long value. The rest of the contents of the blob is ignored. *Stringorblob* can also be an Any variable containing a string or blob.

Return value

Long. Returns the value of *stringorblob* as a long if it succeeds and 0 if *stringorblob* is not a valid PowerScript number or if it is an incompatible datatype. If *stringorblob* is null, Long returns null.

Usage

To distinguish between a string whose value is the number 0 and a string whose value is not a number, use the IsNumber function before calling the Long function.

Examples

This statement returns 2167899876 as a long:

```
Long("2167899876")
```

This statement (PocketBuilder only) returns a decimal value for a hexadecimal string :

```
Long("0xC") // Returns 12.
```

After assigning blob data from the database to *lb_blob*, the following example obtains the long value stored at position 20 in the blob:

```
long lb_num
lb_num = Long(BlobMid(lb_blob, 20, 4))
```

For an example of assigning and extracting values from a blob, see Real.

See also

Dec
Double
Integer
Real
Long method for DataWindows in the *DataWindow Reference*

Lower

Description

Converts all the characters in a string to lowercase.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

Lower (*string*)

Argument	Description
string	The string you want to convert to lowercase letters

Return value

String. Returns *string* with uppercase letters changed to lowercase if it succeeds and the empty string ("") if an error occurs. If *string* is null, Lower returns null.

Examples

This statement returns `babe ruth`:

```
Lower("Babe Ruth")
```

See also

Upper
Lower method for DataWindows in the *DataWindow Reference*

LowerBound

Description

Obtains the lower bound of a dimension of an array.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

LowerBound (*array* {, *n* })

Argument	Description
array	The name of the array for which you want the lower bound of a dimension
n (optional)	The number of the dimension for which you want the lower bound. The default is 1

Return value

Long. Returns the lower bound of dimension *n* of *array* and -1 if *n* is greater than the number of dimensions of the array. If any argument's value is null, LowerBound returns null.

Usage

For variable-size arrays, memory is allocated for the array when you assign values to it. Before you assign values, the lower bound is 1 and the upper bound is 0.

Examples

The following statements illustrate the values LowerBound reports for fixed-size arrays and for variable-size arrays before and after memory has been allocated:

```
integer a[5], b[2,5]
LowerBound(a)    // Returns 1
LowerBound(a, 1) // Returns 1
LowerBound(a, 2) // Returns -1, a has only 1 dim
LowerBound(b, 2) // Returns 1

integer c[ ]
LowerBound(c)    // Returns 1
c[50] = 900
LowerBound(c)    // Returns 1

integer d[-10 to 50]
LowerBound(d)    // Returns  - 10
```

See also

UpperBound

mailAddress

Description — Updates the mailRecipient array for a mail message.

PocketBuilder	✗
PowerBuilder	✓

Applies to — mailSession object

Syntax — *mailsession*.**mailAddress** ({ *mailmessage* })

Return value — mailReturnCode. Returns one of the following values:

mailReturnSuccess!
mailReturnFailure!
mailReturnInsufficientMemory!
mailReturnUserAbort!

mailDeleteMessage

Description — Deletes a mail message from the user's electronic mail inbox.

PocketBuilder	✗
PowerBuilder	✓

Applies to — mailSession object

Syntax — *mailsession*.**mailDeleteMessage** (*messageid*)

Return value — mailReturnCode. Returns one of the following values:

mailReturnSuccess!
mailReturnFailure!
mailReturnInsufficientMemory!
mailReturnInvalidMessage!
mailReturnUserAbort!

mailGetMessages

Description

Populates the messageID array of a mailSession object with the message IDs in the user's inbox.

PocketBuilder	✗
PowerBuilder	✓

Applies to

mailSession object

Syntax

mailsession.**mailGetMessages** ({ *messagetype*, } { *unreadonly* })

Return value

mailReturnCode. Returns one of the following values:

mailReturnSuccess!
mailReturnFailure!
mailReturnInsufficientMemory!
mailReturnNoMessages!
mailReturnUserAbort!

mailHandle

Description

Obtains the handle of a mailSession object.

PocketBuilder	✗
PowerBuilder	✓

Applies to

mailSession object

Syntax

mailsession.**mailHandle** ( )

Return value

UnsignedLong. Returns the internal handle of the mail session object. If *mailsession* is null, mailHandle returns null.

mailLogoff

Description

Ends the mail session, breaking the connection between the PocketBuilder application and mail. If the mail application was already running when PocketBuilder began the mail session, it is left in the same state.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

mailSession object

Syntax

mailsession.**mailLogoff** ( )

Argument	Description
mailsession	A mailSession object identifying the session from which you want to log off

Return value

mailReturnCode. Returns one of the following values:

mailReturnSuccess!
mailReturnFailure!
mailReturnInsufficientMemory!

Usage

To release the memory used by the mailSession object, use the DESTROY keyword after ending the mail session.

Before calling mail functions, you must declare and create a mailSession object and call mailLogon to establish a mail session.

Examples

This statement terminates the current mail session:

```
current_session. mailLogoff()
DESTROY current_session
```

See also

mailLogon

mailLogon

Description

Establishes a mail session for the PocketBuilder application. The PocketBuilder application can start a new session or join an existing session.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

mailSession object

Syntax

mailsession.**mailLogon** ({ *profile*, *password* } {, *logonoption* })

Argument	Description
mailsession	A mailSession object identifying the session you want to logon to.
profile (optional)	A string whose value is the user's mail system profile or user ID.
password (optional)	A string whose value is the user's mail system password.
logonoption (optional)	A value of the mailLogonOption enumerated datatype specifying the logon options: • mailNewSession! — Starts a new mail session, whether or not the mail application is already running • mailDownLoad! — Forces the mail application to download any new messages from the server to the user's inbox. Starts a new mail session only if the mail application is not running • mailNewSessionWithDownLoad! — Starts a new mail session and forces new messages to be downloaded from the server to the user's inbox The default is to use an existing session if possible and not to force new messages to be downloaded. This is the only valid option for PocketBuilder.

Return value

mailReturnCode. Returns one of the following values:

mailReturnSuccess!
mailReturnLoginFailure!
mailReturnInsufficientMemory!
mailReturnTooManySessions!
mailReturnUserAbort!

If any argument's value is null, mailLogon returns null.

Usage

If you do not direct mailLogon to start a new session and the mail application is already running on the user’s computer, then the PocketBuilder mail session attaches to the existing session. A profile and password are not necessary.

When mailLogon establishes a new session, then the mail system’s dialog box prompts for the profile and password if the script does not supply them.

The download option forces the mail server to download the latest messages to the user’s inbox. This ensures that the inbox is up to date; it does not make the messages available to PocketBuilder.

Before calling mailLogon, you must declare and create a mailSession object.

Examples

In this example, the mailSession object *new_session* is an instance variable of the window. The window’s Open event script allocates memory for the mailSession object and logs on. During the logon process, the mail application displays a dialog box prompting for the profile and password:

```
new_session = CREATE mailSession
new_session.mailLogon(mailNewSession!)
```

This example establishes a new mail session and makes the user’s inbox up to date. The user wo not be prompted for an ID and password because user information is provided. Here the mailSession object is a local variable:

```
mailSession new_session
new_session = CREATE mailSession
new_session.mailLogon("jpl", "hotstuff", &
    mailNewSessionWithDownLoad!)
```

See also

mailLogoff

mailReadMessage

Description

Opens a mail message whose ID is stored in the mail session’s message array. You can choose to read the entire message or the envelope (sender, date received, and so on) only. If a message has attachments, they are stored in a temporary file. You can also choose to have the message text written to in a temporary file.

PocketBuilder	✗
PowerBuilder	✓

Applies to

mailSession object

Syntax	*mailsession*.**mailReadMessage** (*messageid*, *mailmessage*, *readoption*, *mark*)
Return value	MailReturnCode. Returns one of the following values: mailReturnSuccess! mailReturnFailure! mailReturnInsufficientMemory! If any argument's value is null, mailReadMessage returns null.

mailRecipientDetails

Description

Displays a dialog box with the specified recipient's address information.

PocketBuilder	✗
PowerBuilder	✓

Applies to

mailSession object

Syntax

mailsession.**mailRecipientDetails** (*mailrecipient* {, *allowupdates* })

Return value

mailReturnCode. Returns one of the following values:

mailReturnSuccess!
mailReturnFailure!
mailReturnInsufficientMemory!
mailReturnUnknownRecipient!
mailReturnUserAbort!

If any argument's value is null, mailRecipientDetails returns null.

mailResolveRecipient

Description

Obtains a valid e-mail address based on a partial or full user name and optionally updates information in the system's address list if the user has privileges to do so.

PocketBuilder	✗
PowerBuilder	✓

Applies to

mailSession object

Syntax *mailsession*.**mailResolveRecipient** (*recipient* {, *allowupdates* })

Return value mailReturnCode. Returns one of the following values:

mailReturnSuccess!
mailReturnFailure!
mailReturnInsufficientMemory!
mailReturnUserAbort!

If any argument's value is null, mailResolveRecipient returns null.

mailSaveMessage

Description Creates a new message in the user's inbox or replaces an existing message.

PocketBuilder	✗
PowerBuilder	✓

Applies to mailSession object

Syntax *mailsession*.**mailSaveMessage** (*messageid*, *mailmessage*)

Return value mailReturnCode. Returns one of the following values:

mailReturnSuccess!
mailReturnFailure!
mailReturnInsufficientMemory!
mailReturnInvalidMessage!
mailReturnUserAbort!
mailReturnDiskFull!

If any argument's value is null, mailSaveMessage returns null.

mailSend

Description Sends a mail message. If no message information is supplied, the mail system provides a dialog box for entering it before sending the message.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

mailSession object

Syntax

mailsession.**mailSend** ({ *mailmessage* })

Argument	Description
mailsession	A mailSession object identifying the session in which you want to send the mail message
mailmessage (optional)	A mailMessage structure

Return value

mailReturnCode. Returns one of the following values:

mailReturnSuccess!
mailReturnFailure!
mailReturnInsufficientMemory!
mailReturnLoginFailure!
mailReturnUserAbort!
mailReturnDiskFull!
mailReturnTooManySessions!
mailReturnTooManyFiles!
mailReturnTooManyRecipients!
mailReturnUnknownRecipient!
mailReturnAttachmentNotFound!

If any argument's value is null, mailSend returns null.

Usage

Before calling mail functions, you must declare and create a mailSession object and call mailLogon to establish a mail session.

For mailSend, mailOriginator! is not a valid value for the Recipient property of the *mailMessage* object. The valid values are mailto!, mailcc!, and mailbcc!. To specify that the sender receive a copy of the message, use mailcc!.

Examples

These statements create a mail session, send a message, and then log off the mail system and destroy the mail session object:

```
mailSession mSes
mailReturnCode mRet
mailMessage mMsg

// Create a mail session
mSes = create mailSession

// Log on to the session
mRet = mSes.mailLogon(mailNewSession!)
```

```
IF mRet <> mailReturnSuccess! THEN
    MessageBox("Mail", 'Logon failed.')
    RETURN
END IF

// Populate the mailMessage structure
mMsg.Subject = mle_subject.Text
mMsg.NoteText = 'Luncheon at 12:15'
mMsg.Recipient[1].name = 'Smith, John'
mMsg.Recipient[2].name = 'Shaw, Sue'

// Send the mail
mRet = mSes.mailSend(mMsg)

IF mRet <> mailReturnSuccess! THEN
    MessageBox("Mail Send", 'Mail not sent')
    RETURN
END IF

mSes.mailLogoff()
DESTROY mSes
```

See also

mailReadMessage
mailResolveRecipient

MakeCall

Description

Places a call using the properties of the PhoneCall object.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✗

Applies to

PhoneCall objects

Syntax

objectname.MakeCall ( )

Argument	Description
objectname	The name of the PhoneCall object on which the call will be made

Return value

Integer. Returns 1 for success and a negative value if an error occurs.

Examples

The following example gets a phone number and name from single-line edit boxes, sets properties of the pcall_1 object, and makes a call:

```
// Global variable: Long g_phInit = 0
integer li_ret
if ( g_phInit > 0) then
   pcall_1.VoiceCall = true
   pcall_1.PhoneNumber = sle_number.text
   pcall_1.CalledParty = sle_name.text
   li_ret = pcall_1.MakeCall()
else
   sle_1.text = "Call not initialized"
end if
```

See also

AcceptCall
AllowReceivingCalls
DropCall
SetHold
SetMute
SetRingTone

Match

Description

Determines whether a string's value contains a particular pattern of characters.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

Match (*string*, *textpattern*)

Argument	Description
string	The string in which you want to look for a pattern of characters
textpattern	A string whose value is the text pattern

Return value

Boolean. Returns true if *string* matches *textpattern* and false if it does not. Match also returns false if either argument has not been assigned a value or the pattern is invalid. If any argument's value is null, Match returns null.

Usage

Match enables you to evaluate whether a string contains a general pattern of characters. To find out whether a string contains a specific substring, use the Pos function.

Textpattern is similar to a regular expression. It consists of metacharacters, which have special meaning, and ordinary characters, which match themselves. You can specify that the string begin or end with one or more characters from a set, or that it contain any characters except those in a set.

A text pattern consists of metacharacters, which have special meaning in the match string, and nonmetacharacters, which match the characters themselves.The following tables explain the meaning and use of these metacharacters.

Table 10-6: Metacharacters used by Match function

Metacharacter	Meaning	Example
Caret (^)	Matches the beginning of a string	^C matches C at the beginning of a string.
Dollar sign ($)	Matches the end of a string	s$ matches s at the end of a string.
Period (.)	Matches any character	. . . matches three consecutive characters.
Backslash (\)	Removes the following metacharacter’s special characteristics so that it matches itself	\$ matches $.
Character class (a group of characters enclosed in square brackets ([ ]))	Matches any of the enclosed characters	[AEIOU] matches A, E, I, O, or U. You can use hyphens to abbreviate ranges of characters in a character class. For example, [A-Za-z] matches any letter.
Complemented character class (first character inside the brackets is a caret)	Matches any character not in the group following the caret	[^0-9] matches any character except a digit, and [^A-Za-z] matches any character except a letter.

The metacharacters asterisk (*), plus (+), and question mark (?) are unary operators that are used to specify repetitions in a regular expression:

Table 10-7: Unary operators used as metacharacters by Match function

Metacharacter	Meaning	Example
* (asterisk)	Indicates zero or more occurrences	A* matches zero or more As (no As, A, AA, AAA, and so on)
+ (plus)	Indicates one or more occurrences	A+ matches one A or more than one A (A, AAA, and so on)
? (question mark)	Indicates zero or one occurrence	A? matches an empty string ("") or A

Sample patterns The following table shows various text patterns and sample text that matches each pattern:

Table 10-8: Text pattern examples for Match function

This pattern	Matches
AB	Any string that contains AB; for example, ABA, DEABC, graphAB_one
B*	Any string that contains 0 or more Bs; for example, AC, B, BB, BBB, ABBBC, and so on
AB*C	Any string containing the pattern AC or ABC or ABBC, and so on (0 or more Bs)
AB+C	Any string containing the pattern ABC or ABBC or ABBBC, and so on (1 or more Bs)
ABB*C	Any string containing the pattern ABC or ABBC or ABBBC, and so on (1 B plus 0 or more Bs)
^AB	Any string starting with AB
AB?C	Any string containing the pattern AC or ABC (0 or 1 B)
^[ABC]	Any string starting with A, B, or C
[^ABC]	A string containing any characters other than A, B, or C
^[^abc]	A string that begins with any character except a, b, or c
^[^a-z]$	Any single-character string that is not a lowercase letter (^ and $ indicate the beginning and end of the string)
[A-Z]+	Any string with one or more uppercase letters
^[0-9]+$	Any string consisting only of digits
^[0-9][0-9][0-9]$	Any string consisting of exactly three digits
^([0-9][0-9][0-9])$	Any consisting of exactly three digits enclosed in parentheses

Examples

This statement returns true if the text in sle_ID begins with one or more uppercase or lowercase letters (^ at the beginning of the pattern means that the beginning of the string must match the characters that follow):

```
Match(sle_ID.Text, "^[A-Za-z]")
```

This statement returns false if the text in sle_ID contains any digits (^ inside a bracket is a complement operator):

```
Match(sle_ID.Text, "[^0-9]")
```

This statement returns true if the text in sle_ID contains one uppercase letter:

```
Match(sle_ID.Text, "[A-Z]")
```

This statement returns true if the text in sle_ID contains one or more uppercase letters (+ indicates one or more occurrences of the pattern):

```
Match(sle_ID.Text, "[A-Z]+")
```

This statement returns false if the text in sle_ID contains anything other than two digits followed by a letter (^ and $ indicate the beginning and end of the string):

```
Match(sle_ID.Text, "^[0-9][0-9][A-Za-z]$")
```

See also

Pos
Match method for DataWindows in the *DataWindow Reference*

MatchW

Description

Determines whether a string's value contains a particular pattern of characters.

PocketBuilder	✗
PowerBuilder	✓

Obsolete function
MatchW is an obsolete function. It has the same behavior as Match.

Syntax

MatchW (*string*, *textpattern*)

Return value

Boolean. Returns true if *string* matches *textpattern* and false if it does not.

Max

Description

Determines the larger of two numbers.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

Max (*x*, *y*)

Argument	Description
x	The number to which you want to compare *y*
y	The number to which you want to compare *x*

Return value

The datatype of *x* or *y*, whichever datatype is more precise. If any argument's value is null, Max returns null.

Usage

If either of the values being compared is null, Max returns null.

Examples

This statement returns 7:

```
Max(4,7)
```

This statement returns -4:

```
Max(- 4, - 7)
```

This statement returns 8.2, a decimal value:

```
Max(8.2, 4)
```

See also

Min
Max method for DataWindows in the *DataWindow Reference*

MaxFARRequested

Description

Sets or retrieves the maximum acceptable value for a false acceptance rate.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✗

Applies to

BiometricScanner objects

Function availability
This function is not used by the HPBiometricScanner object implementation of the BiometricScanner base class.

Syntax

Integer *scanner.*FARRequested ({*maxFAR* })

Argument	Description
scanner	The scanner object associated with the device you want to use to complete a scan
maxFAR	Integer value that you use to set the maximum acceptable FAR

Return value

Integer. For the syntax without the *maxFAR* argument, returns the current value for the maximum acceptable FAR.

For the syntax with the *maxFAR* argument, returns 1 for success or a negative values if an error occurs.

Usage

Together with the maximum false rejection rate, setting the maximum FAR is how you set threshholds for returning scores in the verification stage of a scan operation. If you assign -1 as the value for the *maxFAR* argument, the manufacturer's default setting is used for the maximum FAR value.

Examples

The following example retrieves the maximum FAR to a local variable:

```
li_rtn = l_bioscanner.FARRequested( )
```

See also

FARPrecedence
MaxFRRRequested

MaxFRRRequested

Description

Sets or retrieves the maximum acceptable value for a false rejection rate.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✗

Applies to

BiometricScanner objects

Function availability
This function is not used by the HPBiometricScanner object implementation of the BiometricScanner base class.

Syntax

Integer *scanner*.FRRRequested ({*maxFRR* })

Argument	Description
scanner	The scanner object associated with the device you want to use to complete a scan
maxFRR	Integer value that you use to set the maximum acceptable FRR

Return value

Integer. For the syntax without the *maxFRR* argument, returns the current value for the maximum acceptable FRR.

For the syntax with the *maxFRR* argument, returns 1 for success or a negative values if an error occurs.

Usage

Together with the maximum false acceptance rate, setting the maximum FRR is how you set threshholds for returning scores in the verification stage of a scan operation. If you assign -1 as the value for the *maxFRR* argument, the manufacturer's default setting is used for the maximum FRR value.

Examples

The following example retrieves the maximum FRR to a local variable:

```
li_rtn = l_bioscanner.FRRRequested( )
```

See also

FARPrecedence
MaxFRRRequested

MemberDelete

Description

Deletes a member from an OLE object in a storage. The member can be another OLE object (a substorage) or a stream.

PocketBuilder	✗
PowerBuilder	✓

Applies to

OLEStorage objects

Syntax

olestorage.**MemberDelete** (*membername*)

Return value

Integer. Returns 0 if it succeeds and a negative number if an error occurs.

MemberExists

Description

Determines whether the named member is part of an OLE object in a storage. The member can be another OLE object (a substorage) or a stream.

PocketBuilder	✗
PowerBuilder	✓

Applies to

OLEStorage objects

Syntax

olestorage.**MemberExists** (*membername*, *exists*)

Return value

Integer. Returns 0 if it succeeds and a negative number if an error occurs.

MemberRename

Description

Renames a member in an OLE storage. The member can be another OLE object (a substorage) or a stream.

PocketBuilder	✗
PowerBuilder	✓

Applies to

OLEStorage objects

Syntax

olestorage.**MemberRename** (*membername*, *newname*)

Return value

Integer. Returns 0 if it succeeds and a negative number if an error occurs.

MessageBox

Description

Displays a system MessageBox with the title, text, icon, and buttons you specify.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

MessageBox (*title*, *text* {, *icon* {, *button* {, *default* } } })

Argument	Description
title	A string specifying the title of the message box, which appears in the box's title bar.
text	The text you want to display in the message box. The text can be a numeric datatype, a string, or a boolean value.
icon (optional)	A value of the Icon enumerated datatype indicating the icon you want to display on the left side of the message box. Values are: • Information! (Default) • StopSign! • Exclamation! • Question! • None!

Argument	Description
button (optional)	A value of the Button enumerated datatype indicating the set of CommandButtons you want to display at the bottom of the message box. The buttons are numbered in the order listed in the enumerated datatype. Values are: • OK! — (Default) OK button • OKCancel! — OK and Cancel buttons • YesNo! — Yes and No buttons • YesNoCancel! — Yes, No, and Cancel buttons • RetryCancel! — Retry and Cancel buttons • AbortRetryIgnore! — Abort, Retry, and Ignore buttons
default (optional)	The number of the button you want to be the default button. The default is 1. If you specify a number larger than the number of buttons displayed, MessageBox uses the default.

Return value

Integer. Returns the number of the selected button (1, 2, or 3) if it succeeds and -1 if an error occurs. If any argument's value is null, MessageBox returns null.

Usage

If the value of *title* or *text* is null, the MessageBox does not display. Unless you specify otherwise, PocketBuilder continues executing the script when the user clicks the button or presses enter, which is appropriate when the MessageBox has one button. If the box has multiple buttons, you will need to include code in the script that checks the return value and takes an appropriate action.

Before continuing with the current application, the user must respond to the MessageBox. However, the user can switch to another application without responding to the MessageBox.

When *MessageBox* does not work

Controls capture the mouse in order to perform certain operations. For instance, CommandButtons capture the mouse during mouse clicks, Edit controls capture for text selection, and scrollbars capture during scrolling. If a MessageBox is invoked while the mouse is captured, unexpected results can occur.

Because MessageBox grabs focus, you should not use it when focus is changing, such as in a LoseFocus event. Instead, you might display a message in the window's title or a MultiLineEdit.

MessageBox also causes confusing behavior when called after PrintOpen. For details, see PrintOpen.

Examples

This statement displays a MessageBox with the title Greeting, the text Hello User, the default icon (Information!), and the default button (the OK button):

```
MessageBox("Greeting", "Hello User")
```

The following statements display a MessageBox titled Result and containing the result of a function, the Exclamation icon, and the OK and Cancel buttons (the Cancel button is the default):

```
integer Net
long Distance = 3.457

Net = MessageBox("Result", Abs(Distance), &
    Exclamation!, OKCancel!, 2)
IF Net = 1 THEN
 ... // Process OK.
ELSE
 ... // Process CANCEL.
END IF
```

Mid

Description

Obtains a specified number of characters from a specified position in a string.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

Mid (*string*, *start* {, *length* })

Argument	Description
string	The string from which you want characters returned.
start	A long specifying the position of the first character you want returned. (The position of the first character of the string is 1).
length (optional)	A long whose value is the number of characters you want returned. If you do not enter *length* or if *length* is greater than the number of characters to the right of *start*, Mid returns the remaining characters in the string.

Return value

String. Returns characters specified in *length* of *string* starting at character *start*. If *start* is greater than the number of characters in *string*, the Mid function returns the empty string (""). If *length* is greater than the number of characters remaining after the *start* character, Mid returns the remaining characters. The return string is not filled with spaces to make it the specified length. If any argument's value is null, Mid returns null.

Usage

To search a string for the position of the substring that you want to extract, use the Pos function. Use the return value for the *start* argument of Mid. To extract a specified number of characters from the beginning or end of a string, use the Left or the Right function.

Examples

This statement returns RUTH:

```
Mid("BABE RUTH", 5, 5)
```

This statement returns "":

```
Mid("BABE RUTH", 40, 5)
```

This statement returns BE RUTH:

```
Mid("BABE RUTH", 3)
```

These statements store the characters in the SingleLineEdit sle_address from the 40th character to the end in *ls_address_extra*:

```
string ls_address_extra
ls_address_extra = Mid(sle_address.Text, 40)
```

The following user-defined function, called str_to_int_array, converts a string into an array of integers. Each integer in the array will contain two characters (one characters as the high byte (ASCII value * 256) and the second character as the low byte). The function arguments are *str*, a string passed by value, and *iarr*, an integer array passed by reference. The length of the array is initialized before the function is called. If the integer array is longer than the string, the script stores spaces. If the string is longer, the script ignores the extra characters.

To call the function, use code like the following:

```
int rtn
iarr[20]=0// Initialize the array, if necessary
rtn = str_to_int_array("This is a test.", iarr)
```

The str_to_int_array function is:

```
long stringlen, arraylen, i
string char1, char2
```

```
// Get the string and array lengths
arraylen = UpperBound(iarr)
stringlen = Len(str)

// Loop through the array
FOR i = 1 to arraylen
    IF (i*2 <= stringlen) THEN
      // Get two chars from str
      char1 = Mid(str, i*2, 1)
      char2 = Mid(str, i*2 - 1, 1)

    ELSEIF (i*2 - 1 <= stringlen) THEN
      // Get the last char
      char1 = " "
      char2 = Mid(str, i*2 - 1, 1)

    ELSE
      // Use spaces if beyond the end of str
      char1 = " "
      char2 = " "
    END IF

    iarr[i] = Asc(char1) * 256 + Asc(char2)
NEXT
RETURN 1
```

For sample code that converts the integer array back to a string, see Asc.

See also

Asc
Left
Pos
Right
UpperBound
Mid method for DataWindows in the *DataWindow Reference*

MidW

Description

Obtains a specified number of characters from a specified position in a string.

PocketBuilder	✗
PowerBuilder	✓

Obsolete function
MidW is an obsolete function. It has the same behavior as Mid.

Syntax **MidW** (*string*, *start* {, *length* })

Return value String. Returns characters specified in *length* of *string* starting at character *start*.

Min

Description Determines the smaller of two numbers.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax **Min** (*x*, *y*)

Argument	Description
x	The number to which you want to compare *y*
y	The number to which you want to compare *x*

Return value The datatype of *x* or *y*, whichever datatype is more precise. If any argument's value is null, Min returns null.

Usage If either of the values being compared is null, Min returns null.

Examples This statement returns 4:

```
Min(4,7)
```

This statement returns -7:

```
Min(- 4, - 7)
```

This statement returns 3.0, a decimal value:

```
Min(9.2,3.0)
```

See also Max
Min method for DataWindows in the *DataWindow Reference*

Minute

Description

Obtains the number of minutes in the minutes portion of a time value.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

Minute (*time*)

Argument	Description
time	The time value from which you want the minutes

Return value

Integer. Returns the minutes portion of *time* (00 to 59). If *time* is null, Minute returns null.

Examples

This statement returns 1:

```
Minute(19:01:31)
```

See also

Hour
Second
Minute method for DataWindows in the *DataWindow Reference*

Mod

Description

Obtains the remainder (modulus) of a division operation.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

Mod (*x*, *y*)

Argument	Description
x	The number you want to divide by *y*
y	The number you want to divide into *x*

Return value

The datatype of *x* or *y*, whichever datatype is more precise. If any argument's value is null, Mod returns null.

Examples

This statement returns 2:

```
Mod(20, 6)
```

This statement returns 1.5:

```
Mod(25.5, 4)
```

This statement returns 2.5:

```
Mod(25, 4.5)
```

See also

Mod method for DataWindows in the *DataWindow Reference*

ModifyData

Changes the value of a data point in a series on a graph. There are two syntaxes depending on the type of graph.

To modify a data point in	Use
All graph types except scatter	Syntax 1
Scatter graphs	Syntax 2

Syntax 1 For all graph types except scatter

Description

Changes the value of a data point in a series on a graph. You can specify the data point to be modified by position or by category.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Graph controls in windows and user objects. Does not apply to graphs within DataWindow objects (their data comes directly from the DataWindow).

Syntax

controlname.**ModifyData** (*seriesnumber*, *datapoint*, *datavalue*
{, *categoryvalue* })

Argument	Description
controlname	The name of the graph in which you want to modify data.
seriesnumber	The number of the series in which you want to modify data.
datapoint	The number of the data point for which you want to modify the data.

Argument	Description
datavalue	The new value of the data point. The datatype of *datavalue* is the same as the datatype of the values axis of the graph.
categoryvalue (optional)	The category for *datavalue*. The datatype of *categoryvalue* is the same as the datatype of the category axis of the graph.

Usage

When you specify *categoryvalue*, ModifyData changes the category value at the specified position, as well as the data value. If the name you specify already exists at another position, the data at that position is modified instead and the position in *datapoint* is ignored (the same behavior as InsertData).

When you specify a position of 0, ModifyData always behaves the same as InsertData. For a comparison of AddData, InsertData, and ModifyData, see the Usage section in InsertData.

Examples

These statements change the data for Apr in the series named Costs in the graph gr_product_data:

```
integer SeriesNbr, CategoryNbr
// Get the number of the series.
SeriesNbr = gr_product_data.FindSeries("Costs")
CategoryNbr = gr_product_data.FindCategory("Apr")
gr_product_data.ModifyData(SeriesNbr, &
    CategoryNbr, 1250)
```

See also

AddData
FindCategory
FindSeries
InsertCategory
InsertData

Syntax 2 For scatter graphs

Description

Changes the value of a data point in a series on a graph. You specify the data point by position and provide an x and y value.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Graph controls in windows and user objects. Does not apply to graphs within DataWindow objects (their data comes directly from the DataWindow).

Syntax

controlname.**ModifyData** (*seriesnumber*, *datapoint*, *xvalue*, *yvalue*)

Argument	Description
controlname	The name of the scatter graph in which you want to modify data in a series
seriesnumber	The number that identifies the series in which you want to modify data
datapoint	The number of the data point for which you want to modify data
xvalue	The new x value of the data you want to modify
yvalue	The new y value of the data you want to modify

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is null, ModifyData returns null.

Usage

For scatter graphs, there are no categories. You specify the position in the series whose data you want to modify and provide the x and y values for the data.

Examples

These statements modify the data point 9 in the series named Test One in the scatter graph gr_product_data:

```
integer SeriesNbr
SeriesNbr = gr_product.FindSeries("Test One")
gr_product_data.ModifyData(SeriesNbr, &
    9, 4.55, 86.38)
```

See also

AddData
FindSeries

Month

Description

Determines the month of a date value.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

Month (*date*)

Argument	Description
date	The date from which you want the month

Return value

Integer. Returns an integer (1 to 12) whose value is the month portion of *date*. If *date* is null, Month returns null.

Examples

This statement returns 1:

```
Month(1994-01-31)
```

These statements store in *start_month* the month entered in the SingleLineEdit sle_start_date:

```
integer start_month
start_month = Month(date(sle_start_date.Text))
```

See also

Day
Date
Year
Month method for DataWindows in the *DataWindow Reference*

Move

Description

Moves a control or object to another position relative to its parent window, or for some window objects, relative to the screen.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Any object or control

Syntax

objectname.**Move** (*x*, *y*)

Argument	Description
objectname	The name of the object or control you want to move to a new location
x	The x coordinate of the new location in PowerBuilder units
y	The y coordinate of the new location in PowerBuilder units

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs or if *objectname* is a maximized window. If any argument's value is null, Move returns null.

Usage

The x and y coordinates you specify are the new coordinates of the upper-left corner of the object or control. If the shape of the object or control is not rectangular (such as, a RadioButton or Oval), x and y are the coordinates of the upper-left corner of the box enclosing it.

When you move controls, drawing objects, and child windows, the coordinates you specify are relative to the upper-left corner of the parent window. When you use Move to position main, pop-up, and response windows, the coordinates you specify are relative to the upper-left corner of the display screen.

Move does not move a maximized sheet or window. If the window is maximized, Move returns –1.

You can use Move to move a line control but the results are unpredictable because the line has multiple x and y coordinates.

You can specify coordinates outside the frame of the parent window or screen, which effectively makes the object or control invisible.

To draw the image of a Picture control at a particular position, without actually moving the control, use the Draw function.

The Move function changes the X and Y properties of the moved object.

Equivalent syntax The syntax below directly sets the X and Y properties of an object or control. Although the result is equivalent to using the Move function, it causes PocketBuilder to redraw *objectname* twice, first at the new location of X and then at the new X and Y location:

objectname.**X** = *x*

objectname.**Y** = *y*

These statements cause PocketBuilder to redraw gb_box1 twice:

```
gb_box1.X = 150
gb_box1.Y = 200
```

This statement has the same result but redraws gb_box1 once:

```
gb_box1.Move(150,200)
```

Examples

This statement changes the X and Y properties of gb_box1 to 150 and 200, respectively, and moves gb_box1 to the new location:

```
gb_box1.Move(150, 200)
```

This statement moves the picture p_Train2 next to the picture p_Train1:

```
P_Train2.Move(P_Train1.X + P_Train1.Width, &
    P_Train1.Y)
```

MoveTab

Description

Moves a tab page to another position in a Tab control, changing its index number.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✓

Applies to

Tab controls

Syntax

tabcontrolname.**MoveTab** (*source*, *destination*)

Argument	Description
tabcontrolname	The name of the Tab control containing the tab you want to move.
source	An integer whose value is the index of the tab you want to move.
destination	An integer whose value is the index of the destination tab before which *source* is moved. If *destination* is 0 or greater than the number of tabs, *source* is moved to the end.

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs.

Usage

MoveTab also reorders the tab pages in the Tab control's Control array (which is a property that lists the tab pages within the Tab control) to match the new tab order.

Examples

This example moves the first tab to the end:

```
tab_1.MoveTab(1, 0)
```

This example move the fourth tab to the first position:

```
tab_1.MoveTab(4, 1)
```

This example move the fourth tab to the third position:

```
tab_1.MoveTab(4, 3)
```

See also

OpenTab
SelectTab

_Narrow

Description

Converts a CORBA object reference from a general supertype to a more specific subtype.

This function is used by PowerBuilder clients connecting to EAServer.

PocketBuilder	✗
PowerBuilder	✓

Applies to

CORBAObject objects

Syntax

corbaobject.**_Narrow** (*newremoteobject, classname*)

Return value

Long. Returns 0 if it succeeds and a negative number if an error occurs.

NextActivity

Description

Provides the next activity in a trace file.

PocketBuilder on Desktop	✓
PocketBuilder on Pocket PC	✗
PocketBuilder on Smartphone	✗
PowerBuilder	✓

Applies to

TraceFile objects

Syntax

instancename.**NextActivity** ( )

Argument	Description
instancename	Instance name of the TraceFile object

Return value

TraceActivityNode

Usage

You use the NextActivity function to read the next activity in a trace file. The activity is returned as a TraceActivityNode object. If there are no more activities or if the file is not open, an invalid object is returned. You can then use the LastError property of the TraceFile object to determine what kind of error occurred.

To use this function, you must have previously opened the trace file with the Open function. You use the NextActivity and Open functions as well as the other properties and functions provided by the TraceFile object to access the contents of a trace file directly. For example, you would use these functions if you want to perform your own analysis of the tracing data instead of using the available modeling objects.

Examples

This example opens a trace file and then uses a user-defined function called of_dumpactivitynode to report the appropriate information for each activity depending on its activity type:

```
String ls_filename, ls_line
TraceFile ltf_file
TraceActivityNode ltan_node

ls_filename = sle_filename.text
ltf_file = CREATE TraceFile
ltf_file.Open(ls_filename)

ls_line = "CollectionTime = " + &
    String(ltf_file.CollectionTime) + "~r~n" + &
      "Num Activities = " + &
        String(ltf_file.NumberOfActivities) + "~r~n
mle_output.text = ls_line

ltan_node = ltf_file.NextActivity()
DO WHILE IsValid(ltan_node)
    ls_line = of_dumpactivitynode(ltan_node)
    ltan_node = ltf_file.NextActivity()
    mle_output.text = ls_line
LOOP
```

See also

Open
Close
Reset

Now

Description

Obtains the current time based on the system time of the client machine.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

Now ( )

Return value

Time. Returns the current time based on the system time of the client machine.

Usage

Use Now to compare a time to the system time or to display the system time on the screen. You can use the Timer function to trigger a Timer event which causes Now to refresh the display.

Examples

This statement returns the current system time.

```
Now()
```

This example displays the current time in the StaticText st_time. It keeps the time up-to-date by setting a timer that triggers a Timer event every 60 seconds. Code in the window's Open event displays the initial time and starts the timer. Code in the Timer event displays the time again.

The following code appears in the window's Open event script:

```
st_time.Text = String(Now(), "hh:mm")
Timer(60)
```

A single line in the Timer event script refreshes the time display:

```
st_time.Text = String(Now(), "hh:mm")
```

See also

Today
Now method for DataWindows in the *DataWindow Reference*

ObjectAtPointer

Description	Finds out where the user clicked in a graph. ObjectAtPointer reports the region of the graph under the pointer and stores the associated series and data point numbers in the designated variables.

PocketBuilder	✗
PowerBuilder	✓

Applies to	Graph controls in windows and user objects, and graphs in DataWindow controls
Syntax	*controlname*.**ObjectAtPointer** ({ *graphcontrol*, } *seriesnumber*, *datapoint*)
Return value	grObjectType. Returns a value of the grObjectType enumerated datatype if the user clicks anywhere in the graph (including an empty area) and a null value if the user clicks outside the graph. If any argument's value is null, ObjectAtPointer also returns null.

Object_To_String

Description	Gets the string form of an object. This function is used by PowerBuilder clients connecting to EAServer.

PocketBuilder	✗
PowerBuilder	✓

Applies to	JaguarORB objects
Syntax	*jaguarorb*.**Object_To_String** (*object*)
Return value	String. Returns the string representation of a CORBA object.

OffsetPos

Description

Sets the offset for progress bar controls.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Progress bar controls

Syntax

control.**OffsetPos** (*increment*)

Argument	Description
control	The name of the progress bar control
increment	An integer that is added to the start position of the progress bar control

Return value

Integer. Returns 1 if it succeeds and -1 if there is an error.

Examples

This statement offsets the start position of a horizontal progress bar by 10:

```
HProgressBar.OffsetPos ( 10 )
```

See also

SelectionRange
SetRange
StepIt

Open

Opens a window, connects to a scanner, camera, or GPS device, or opens a file and selects its access mode.

For windows Open displays a window and makes all its properties and controls available to scripts.

To	Use
Open an instance of a particular window datatype	Syntax 1
Allow the application to select the window's datatype when the script is executed	Syntax 2

For BarcodeScanner and BiometricScanner objects Open loads scanner DLLs and connects to scanner firmware.

To	Use
Connect to scanner firmware	Syntax 3

For GPS or SerialGPS objects Open opens a communications channel or provides raw data for use by a GPS object.

To	Use
For the GPS base class	Syntax 4
For the SerialGPS object	Syntax 5

For other objects Open opens a file and selects its access mode.

To	Use
Open a communications channel or initialize a Camera object with a set of raw data	Syntax 6
Open a Short Message Service (SMS) session	Syntax 7
Open a file for reading or writing with the FileDirect object	Syntax 8
Open a trace file	Syntax 9

Syntax 1 For windows of a known datatype

Description

Opens a window object of a known datatype. Open displays the window and makes all its properties and controls available to scripts.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Window objects

Syntax

Open (*windowvar* {, *parent* })

Argument	Description
windowvar	The name of the window you want to display. You can specify a window object defined in the Window painter (which is a window datatype) or a variable of the desired window datatype. Open places a reference to the opened window in *windowvar*.
parent (child and pop-up windows only) (optional)	The window you want make the parent of the child or pop-up window you are opening. If you open a child or pop-up window and omit *parent*, PowerBuilder associates the window being opened with the currently active window.

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is null, Open returns null.

Usage

You must open a window before you can access the properties of the window. If you access the window's properties before you open it, an execution error will occur.

To reference an open window in scripts, use *windowvar*.

Calling Open twice
If you call Syntax 1 of the Open function twice for the same window, PocketBuilder activates the window twice; it does not open two instances of the window.

To open an array of windows where each window has different datatype, use Syntax 2 of Open.

Parent windows for the opened window Generally, if you are opening a child or a pop-up window and specify *parent*, the window identified by *parent* is the parent of the opened window (*windowname* or *windowvar*). When a parent window is closed, all its child and pop-up windows are closed too.

Mouse behavior and response windows
Controls capture the mouse or stylus action in order to perform certain operations. For instance, CommandButtons capture during mouse clicks, edit controls capture for text selection, and scroll bars capture during scrolling. If a response window is opened while the mouse is captured, unexpected results can occur.

Because a response window grabs focus, you should not open it when focus is changing, such as in a LoseFocus event.

Examples

This statement opens an instance of a window named w_employee:

```
Open(w_employee)
```

The following statements open an instance of a window of the type w_employee:

```
w_employee w_to_open
Open(w_to_open)
```

The following code opens an instance of a window of the type child named cw_data and makes w_employee the parent:

```
child cw_data
Open(cw_data, w_employee)
```

The following code opens two windows of type w_emp:

```
w_emp w_e1, w_e2
Open(w_e1)
Open(w_e2)
```

See also

Close
OpenWithParm
Show

Syntax 2 For windows of unknown datatype

Description

Opens a window object when you do not know its datatype until the application is running. Open displays the window and makes all its properties and controls available to scripts.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Window objects

Syntax

Open (*windowvar*, *windowtype* {, *parent* })

Argument	Description
windowvar	A window variable, usually of datatype window. Open places a reference to the opened window in *windowvar*.

Argument	Description
windowtype	A string whose value is the datatype of the window you want to open. The datatype of *windowtype* must be the same or a descendant of *windowvar*.
parent (child and pop-up windows only) (optional)	The window you want to make the parent of the child or pop-up window you are opening. If you open a child or pop-up window and omit *parent*, PowerBuilder associates the window being opened with the currently active window.

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is null, Open returns null.

Usage

You must open a window before you can access the properties of the window. If you access the window's properties before you open it, an execution error will occur.

To reference an open window in scripts, use *windowvar*.

The window object specified in *windowtype* must be the same datatype as *windowvar* (the datatype includes datatypes inherited from it). The datatype of *windowvar* is usually window, from which all windows are inherited, but it can be any ancestor of *windowtype*. If it is not the same type, an execution error will occur.

Use this syntax to open an array of windows when each window in the array will have a different datatype. See the last example, in which the window datatypes are stored in one array and are used for the *windowtype* argument when each window in another array is opened.

Considerations when specifying a window type

When you use Syntax 2, PocketBuilder opens an instance of a window of the datatype specified in *windowtype* and places a reference to this instance in the variable *windowvar*.

If *windowtype* is a descendent window, you can only reference properties, events, functions, or structures that are part of the definition of *windowvar*. For example, if a user event is declared for *windowtype*, you cannot reference it.

The object specified in *windowtype* is not automatically included in your executable application. To include it, you must save it in a PKD file (PocketBuilder dynamic library) that you deliver with your application.

For information about the parent of an opened window, see Syntax 1.

Examples

This example opens a window of the type specified in the string *s_w_name* and stores the reference to the window in the variable *w_to_open*. The SELECT statement retrieves data specifying the window type from the database and stores it in *s_w_name*:

```
window w_to_open
string s_w_name

SELECT next_window INTO  : s_w_name FROM routing_table
WHERE...  ;

Open(w_to_open, s_w_name)
```

This example opens an array of ten windows of the type specified in the string *is_w_emp1* and assigns a title to each window in the array. The string *is_w_emp1* is an instance variable whose value is a window type:

```
integer n
window win_array[10]

FOR n = 1 to 10
      Open(win_array[n], is_w_emp1)
      win_array[n].title = "Window " + string(n)
NEXT
```

The following statements open four windows. The type of each window is stored in the array *w_stock_type*. The window reference from the Open function is assigned to elements in the array *w_stock_win*:

```
window w_stock_win[ ]
string w_stock_type[4]

w_stock_type[1] = "w_stock_wine"
w_stock_type[2] = "w_stock_scotch"
w_stock_type[3] = "w_stock_beer"
w_stock_type[4] = "w_stock_soda"

FOR n = 1 to 4
      Open(w_stock_win[n], w_stock_type[n])
NEXT
```

See also

Close
OpenWithParm
Show

Syntax 3 For BarcodeScanner and BiometricScanner objects

Description

Loads DLLs and connects to scanner firmware.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✗

Applies to

BarcodeScanner and BiometricScanner objects

Syntax

Integer *scanner*.Open ( )

Argument	Description
scanner	The scanner object that you want to open

Return value

Integer. Returns 1 for success or one of the following negative values if an error occurs:

- **-1** Unspecified error
- **-2** Supporting DLL not loaded error
- **-3** Initialization error other than DLL not loaded
- **-4** Error in the passed in arguments
- **-5** Something in the object instance is inconsistent
- **-6** Call to the driver failed
- **-7** Error opening the specific scan device
- **-8** Error in the internal buffer allocation
- **-9** Incorrect scan state for the requested action (typically benign)
- **-10** Low level device error
- **-11** Read is already pending (typically benign)
- **-100** Feature not implemented

Usage

This is typically the first method to call after creation of a scanner object.

Examples

The following example loads scanner DLLs and connects to the scanner device firmware:

```
li_rtn = l_scanner.Open()
```

See also

Close
RetrieveData

Syntax 4 For GPS objects

Description

Opens an ANSI text file containing NMEA sentences for a GPS object and reads the contents into a buffer.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✗

Applies to

GPS objects

Syntax

GPSname.Open ({ *rawdatafile* })

Argument	**Description**
GPSname	Name of the GPS object.
rawdatafile	A string for an ASCII text file containing raw data in NMEA-0183 format to be used by the GPS object.

Return value

Integer. Returns 1 for success and a negative number for any error. The following is a list of possible error codes and their meanings:

-1 General error.

-10 Invalid object. Could occur if the GPS object instance is corrupted.

-11 No RawData. This error is generated when the ConfigParams property is empty and Open is called without a file name argument.

-12 Invalid File. This error is generated on an Open call containing a file name argument when the file does not exist or cannot be opened successfully.

-15 Read Failure. This error is generated on an Open call containing a file name argument when the file cannot be read.

-18 Already Open. An Open request was issued and the object is already open.

Usage

Use this function to populate the fields of the GPS base object.

The optional *rawdatafile* argument is used when the data to be loaded resides in an ANSI text file. The entire data file is read into a buffer for use by the GetFix, GetHeading, and GetSatellitesInView routines. Raw data files must be ANSI text.

Examples

The following lines create a GPS object, retrieve information about the current position fix, and test the validity of the GPSFix object:

```
Gps myGPS
```

```
GPSFix myFix
Integer rc
String errmsg

MyGPS = CREATE GPS
rc = myGPS.Open("c:\data\ConcordMA.txt")
IF rc = 1 THEN
  rc = MyGPS.GetFix(myFix)
  // process fix data
ELSE
    // process error message with user function
   errmsg = uf_display_error("Fix Error", rc)

END IF
```

See also

Close
GetFix
GetHeading
GetSatellitesInView

Syntax 5 For SerialGPS objects

Description

Opens a communications channel for a SerialGPS object and initializes data handlers.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✗

Applies to

SerialGPS objects

Syntax

GPSname.Open ( )

Argument	Description
GPSname	Name of the SerialGPS object.

Return value

Integer. Returns 1 for success and a negative number for any error. The following is a list of possible error codes and their meanings:

-1 General error.

-10 Invalid object. Could occur if the SerialGPS object instance is corrupted.

-15 Read Failure. Unable to read the serial port.

-18 Already Open. An Open request was issued and the object is already open.

Usage

Use this function to open a communications channel for a SerialGPS object and initialize it so that it can be used to obtain GPS information. For the SerialGPS object, you must previously set the SerialPort property and optionally set the ConfigParams proeperty prior to calling this function.

Examples

The following lines create a SerialGPS object, retrieve information about the current position fix, and test the validity of the GPSFix object:

```
SerialGps myGPS
GPSFix myFix
Integer rc

MyGPS = CREATE SerialGPS
rc = myGPS.Open()
IF rc = 1 THEN
  rc = MyGPS.GetFix(myFix)
  IF rc = 1 THEN
    IF myFix.IsFixValid THEN
      // process fix data
    END IF
  ELSE
    // process error message
  END IF
END IF
```

See also

Close
GetFix
GetHeading
GetSatellitesInView

Syntax 6 For Camera objects

Description

Opens a communications channel for a Camera object and initializes data handlers.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✗

Applies to

Camera objects

Syntax

cameraname.Open (*AppWindow*)

Argument	Description
cameraname	Name of the Camera object.
AppWindow	GraphicObject that is required by some camera drivers. Typically you use the name of the main application window.

Return value

Integer. Returns 1 for success, and one of the following negative values if an error occurs:

-1	Unspecified error
-2	Supporting DLL not loaded
-3	Unknown camera type
-4	Other initialization error
-5	Inconsistency in this object instance
-6	Call to the driver or device failed
-7	Unsupported option
-8	Value for option is out of range

Usage

Use this function to open a communications channel for a Camera object and initialize it so that it can be used to capture images.

You must set the camera type before you call Open. You must also set either the port or the folder property of the Camera object, depending on the type of camera device you are using. You can set these properties in the Properties view for a Camera object or in a script. The following table describes the properties you need to set for different devices:

Device	CameraType specifier	Port or Folder property (value)
Windows Mobile 5	1	—
VEO 130S	11	Port (set to "SIO1:")
HP Photosmart	71	Port (set to "SIO1:")
HTC using the IA Camera Wizard	81	Folder (set to the path on the Windows CE device)

Examples

The following code creates a Camera object that interfaces with an HP Photosmart camera:

```
Camera myCamera

myCamera = CREATE Camera
myCamera.Port= "SIO1:"
myCamera.CameraType=71
myCamera.**Open**(w_myphoto_main)
...
```

See also

Close
CaptureImage
GetOption
IsReadyToCapture
SetCaptureImageAttributes
SetOption
SetPreviewImageAttributes

Syntax 7 For SMSSession objects

Description

Opens an SMSSession object.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✗

Applies to

SMSSession objects

Syntax

SMSSessionname.Open (*smsproto*, *msgmodes*)

Argument	Description
SMSSessionname	Name of the SMSSession object.
smsproto	An SMSProtocol structure.
msgmodes	A long value indicating whether the SMSSession object is opened in send or receive mode. Values can be: • **0** send only (retained to support legacy code) • **1** send only • **2** receive only • **3** send and receive

Return value

Integer. Returns 1 for success and a negative number for any error.

Usage

You can send SMS messages from applications running on any supported Windows CE platform, but you can receive SMS messages only in applications running on Windows Mobile platforms. To receive messages, you also need to deploy a shim DLL that installs with PocketBuilder, and you must register the DLL with the operating system where you deploy it.

For more information about receiving SMS messages in a PocketBuilder application, see the chapter on working with native objects and controls in the *Users Guide*.

See also

Close
GetMessageStatus
Send

Syntax 8 For FileDirect objects

Description

Use one of these syntaxes to open a file and select its access mode. Use instead of FileOpen to interface directly with the underlying file system when you want to read from or write to a device connected through BlueTooth or other connection tools. The Open function maps to the Windows CE CreateFile command.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✗

Applies to

FileDirect objects

Syntax

Integer *instancename*.Open (*filename*, *accessmode*, {*sharemode*, *creationdisposition*, *attributes*})

Argument	Description
instancename	Name of the instance of the FileDirect object
filename	A string for the name of the file you want to read from or write to
accessmode	Enumerated value of type stgreadmode. Values can be: • **stgread!** • **stgreadwrite!** • **stgwrite!**
sharemode	Enumerated value of type stgsharemode. Values can be: • **stgdenynone!** • **stgdenyread!** • **stgdenywrite!** • **stgexclusive!**

Argument	Description
creationdisposition	Integer indicating what action to take based on whether or not the designated file exists. Values are: • **1** Creates a new file if file does not exist, but returns an error if it does exist • **2** Creates a new file, overwriting an existing file if necessary • **3** Opens an existing file, but returns an error if the file does not exist • **4** Opens an existing file or creates a new file if the file does not exist • **5** Opens and removes the content of an existing file, but returns an error if the file does not exist
attributes	Integer specifying a handle to a template file that supplies file attributes for the file that you open or create

Return value

Integer. Returns 1 for success and a negative number for any error. Error codes are:

- **-1** Unspecified error
- **-2** File not opened
- **-3** Initialization error
- **-4** Error in the passed in arguments
- **-5** File is read-only
- **-6** File is write-only
- **-7** File is not open
- **-8** Data read but less than expected
- **-9** File already open, but not in share mode

Usage

Use this function to open a file in read or write mode. The FileDirect object supports only the synchronous style of file input or output; further file-related commands cannot be processed until the indicated file is successfully opened or an error in opening the file is caught. The Open function calls the CreateFile method on the device operating system.

Examples

The following example calls the FileDirect user object nvo_fileDirect to open a file, read some data, store the data in a blob variable, and close the file:

```
Integer li_ret, li_AmountRead
Blob lb_data
li_ret = nvo_fileDirect.Open ("MyDoc.txt", stgRead!)
li_ret = nvo_fileDirect.Read (lb_data, 100,
    li_amountRead)
li_ret = nvo_fileDirect.Close ( )
```

See also

Close
Read

Syntax 9 For opening trace files

Description

Opens the specified trace file for reading.

PocketBuilder on Desktop	✓
PocketBuilder on Pocket PC	✗
PocketBuilder on Smartphone	✗
PowerBuilder	✓

Applies to

TraceFile object

Syntax

***instancename*.Open** (*filename*)

Argument	Description
instancename	Instancename of the TraceFile object
filename	A string identifying the name of the trace file you want to read

Return value

ErrorReturn. Returns one of the following values:

- Success!—The function succeeded
- FileAlreadyOpenError!—The specified trace file has already been opened
- FileOpenError!—The trace file can not be opened for reading
- FileInvalidFormatError!—The file does not have the correct format
- EnterpriseOnlyFeature!—This function is supported only in the Enterprise edition of PowerBuilder
- SourcePBLError!—The source libraries cannot be found

Usage

You use this syntax to access the contents of a specified trace file created from a running PocketBuilder application. You can then use the properties and functions provided by the TraceFile object to perform your own analysis of tracing data instead of using the available modeling objects.

Examples

This example opens a trace file:

```
TraceFile ltf_file
String ls_filename

ltf_file = CREATE TraceFile
ltf_file.Open(ls_filename)
...
```

See also

Close
NextActivity
Reset

OpenChannel

Description

Opens a channel to a DDE server application.

PocketBuilder	✗
PowerBuilder	✓

Syntax

OpenChannel (*applname*, *topicname* {, *windowhandle* })

Return value

Long. Returns the handle to the channel (a positive integer) if it succeeds. If an error occurs, OpenChannel returns a negative integer.

OpenSheet

Description

Opens a sheet within an MDI (multiple document interface) frame window and creates a menu item for selecting the sheet on the specified menu.

PocketBuilder	✗
PowerBuilder	✓

Applies to

Window objects

Syntax

OpenSheet (*sheetrefvar* {, *windowtype* }, *mdiframe* {, *position* {, *arrangeopen* } })

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is null, OpenSheet returns null.

OpenSheetWithParm

Description

Opens a sheet within an MDI (multiple document interface) frame window and creates a menu item for selecting the sheet on the specified menu, as OpenSheet does. OpenSheetWithParm also stores a parameter in the system's Message object so that it is accessible to the opened sheet.

PocketBuilder	✗
PowerBuilder	✓

Applies to

Window objects

Syntax

OpenSheetWithParm (*sheetrefvar*, *parameter* {, *windowtype* }, *mdiframe* {, *position* {, *arrangeopen* } })

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is null, OpenSheetWithParm returns null.

OpenTab

Opens a visual user object and makes it a tab page in the specified Tab control and makes all its properties and controls available to scripts.

To open	Use
A user object as a tab page	Syntax 1
A user object as a tab page, allowing the application to select the user object's type during execution	Syntax 2

Syntax 1 For user objects of a known datatype

Description

Opens a custom visual user object of a known datatype as a tab page in a Tab control.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✓

Applies to

Tab controls

Syntax

tabcontrolname.**OpenTab** (*userobjectvar*, *index*)

Argument	Description
tabcontrolname	The name of the Tab control in which you want to open the user object as a tab page.
userobjectvar	The name of the custom visual user object you want to open as a tab page. You can specify a custom visual user object defined in the User Object painter (which is a user object datatype) or a variable of the desired user object datatype. OpenTab places a reference to the opened custom visual user object in *userobjectvar*.
index	The number of the tab before which you want to insert the new tab. If *index* is 0 or greater than the number of tabs, the tab page is inserted at the end.

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is null, OpenTab returns null.

Usage

Use Syntax 1 when you know what user object you want to open. Use Syntax 2 when the application will determine what type of user object to open when the script runs.

The tab page for the user object does not become selected. Scripts for constructor events of the controls on the user object do not run until the tab page is selected.

You must open a user object before you can access the properties of the user object. If you access the user object's properties before you open it, an execution error will occur.

A user object that is part of a Tab control's definition (that is, it was added to the Tab control in the Window painter) does not have to be opened in a script. PocketBuilder opens it when it opens the window containing the Tab control.

OpenTab adds the newly opened user object to the Tab control's Control array, which is a property that lists the tab pages within the Tab control.

Opening the same object twice
If you call Syntax 1 twice to open the same user object, PocketBuilder does open the user object again as another tab page, in contrast to the behavior of Open and OpenUserObject.

Examples

This statement opens an instance of a user object named u_Employee as a tab page in the Tab control tab_1:

```
tab_1.OpenTab(u_Employee, 0)
```

The following statements open an instance of a user object u_to_open as a tab page in the Tab control tab_1. It becomes the first tab in the control:

```
u_employee u_to_open
tab_1.OpenTab(u_to_open, 1)
```

See also

OpenTabWithParm

Syntax 2 For user objects of unknown datatype

Description

Opens a visual user object as a tab page within a Tab control when the datatype of the user object is not known until the script is executed.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✓

Applies to

Tab controls

Syntax

tabcontrolname.**OpenTab** (*userobjectvar*, *userobjecttype*, *index*)

Argument	Description
tabcontrolname	The name of the Tab control in which you want to open the user object as a tab page.
userobjectvar	A variable of datatype UserObject. OpenTab places a reference to the opened user object in *userobjectvar*.
userobjecttype	A string whose value is the name of the user object you want to open. The datatype of *userobjecttype* must be a descendant of *userobjectvar*.
index	The number of the tab before which you want to insert the new tab. If *index* is 0 or greater than the number of tabs, the tab page is inserted at the end

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is null, OpenTab returns null.

Usage

Use Syntax 1 when you know what user object you want to open. Use Syntax 2 when the application will determine what type of user object to open when the script runs.

The tab page for the user object does not become selected. Scripts for Constructor events of the controls on the user object do not run until the tab page is selected.

You must open a user object before you can access the properties of the user object. If you access the user object's properties before you open it, an execution error will occur.

A user object that is part of a Tab control's definition (that is, it was added to the Tab control in the Window painter) does not have to be opened in a script. PocketBuilder opens it when it opens the window containing the Tab control.

OpenTab adds the newly opened user object to the Tab control's Control array, which is a property that lists the tab pages within the Tab control.

Considerations when specifying a user object type

When you use Syntax 2, PocketBuilder opens an instance of a user object of the datatype specified in *userobjecttype* and places a reference to this instance in the variable *userobjectvar*. To refer to the instance in scripts, use *userobjectvar*.

If *userobjecttype* is a descendent user object, you can only refer to properties, events, functions, or structures that are part of the definition of *userobjectvar*. For example, if a user event is declared for *userobjecttype*, you cannot reference it.

The object specified in *userobjecttype* is not automatically included in your executable application. To include it, you must save it in a PKD file (PocketBuilder dynamic library) that you deliver with you application.

Examples

The following example opens a user object as the last tab page in the Tab control tab_1. The user object is of the type specified in the string *s_u_name* and stores the reference to the user object in the variable *u_to_open*:

```
UserObject u_to_open
string s_u_name

s_u_name = sle_user.Text
tab_1.OpenTab(u_to_open, s_u_name, 0)
```

See also

OpenTabWithParm

OpenTabWithParm

Adds a visual user object to the specified window and makes all its properties and controls available to scripts, as OpenTab does. OpenTabWithParm also stores a parameter in the system's Message object so that it is accessible to the opened object.

To open	Use
A user object as a tab page	Syntax 1
A user object as a tab page, allowing the application to select the user object's type during execution	Syntax 2

Syntax 1 For user objects of a known datatype

Description

Opens a custom visual user object of a known datatype as a tab page in a Tab control and stores a parameter in the system's Message object.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✓

Applies to

Tab controls

Syntax

tabcontrolname.**OpenTabWithParm** (*userobjectvar*, *parameter*, *index*)

Argument	Description
tabcontrolname	The name of the Tab control in which you want to open the user object as a tab page.
userobjectvar	The name of the custom visual user object you want to open as a tab page. You can specify a custom visual user object defined in the User Object painter (which is a user object datatype) or a variable of the desired user object datatype. OpenTabWithParm places a reference to the opened custom visual user object in *userobjectvar*.
parameter	The parameter you want to store in the Message object when the user object is opened. *Parameter* must have one of these datatypes: • String • Numeric • PowerObject

Argument	Description
index	The number of the tab before which you want to insert the new tab. If *index* is 0 or greater than the number of tabs, the tab page is inserted at the end.

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is null, OpenTabWithParm returns null.

Usage

The system Message object has three properties for storing data. Depending on the datatype of the parameter specified for OpenTabWithParm, scripts for the opened user object would check one of the following properties.

Message object property	Argument datatype
message.DoubleParm	Numeric
message.PowerObjectParm	PowerObject (PocketBuilder objects, including user-defined structures)
message.StringParm	String

In the opened user object, it is a good idea to access the value passed in the Message object immediately because some other script may use the Message object for another purpose.

Avoiding null object references
When you pass a PowerObject as a parameter, you are passing a reference to the object. The object must exist when you refer to it later or you get a null object reference, which causes an error. For example, if you pass the name of a control on a window that is being closed, that control will not exist when a script accesses the parameter.

See also the usage notes for OpenTab, all of which apply to OpenTabWithParm.

Examples

This statement opens an instance of a user object named u_Employee as a tab page in the Tab control tab_empsettings. It also stores the string `James Newton` in Message.StringParm. The Constructor event script for the user object uses the string parameter as the text of a StaticText control st_empname in the object. The script that opens the tab page has the following statement:

```
tab_empsettings.OpenTabWithParm(u_Employee, &
      "James Newton", 0)
```

The user object's Constructor event script has the following statement:

```
st_empname.Text = Message.StringParm
```

The following statements open an instance of a user object *u_to_open* as the first tab page in the Tab control tab_empsettings and store a number in message.DoubleParm. The last statement selects the tab page:

```
u_employee u_to_open
integer age = 50
tab_1.OpenTabWithParm(u_to_open, age, 1)
tab_1.SelectTab(u_to_open)
```

See also

OpenTab

Syntax 2 For user objects of unknown datatype

Description

Opens a visual user object as a tab page within a Tab control when the datatype of the user object is not known until the script is executed. In addition, OpenTabWithParm stores a parameter in the system's Message object so that it is accessible to the opened object.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✓

Applies to

Tab controls

Syntax

tabcontrolname.**OpenTabWithParm** (*userobjectvar*, *parameter*, *userobjecttype*, *index*)

Argument	Description
tabcontrolname	The name of the Tab control in which you want to open the user object as a tab page.
userobjectvar	A variable of datatype UserObject. OpenTabWithParm places a reference to the opened user object in *userobjectvar*
parameter	The parameter you want to store in the Message object when the user object is opened. *Parameter* must have one of these datatypes: • String • Numeric • PowerObject

Argument	Description
userobjecttype	A string whose value is the datatype of the user object you want to open. The datatype of *userobjecttype* must be a descendant of *userobjectvar*.
index	The number of the tab before which you want to insert the new tab. If *index* is 0 or greater than the number of tabs, the tab page is inserted at the end.

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is null, OpenTabWithParm returns null.

Usage

The system Message object has three properties for storing data. Depending on the datatype of the parameter specified for OpenTabWithParm, scripts for the opened user object would check one of the following properties.

Message object property	Argument datatype
message.DoubleParm	Numeric
message.PowerObjectParm	PowerObject (PocketBuilder objects, including user-defined structures)
message.StringParm	String

In the opened user object, it is a good idea to access the value passed in the Message object immediately because some other script may use the Message object for another purpose.

Avoiding null object references
When you pass a PowerObject as a parameter, you are passing a reference to the object. The object must exist when you refer to it later or you will get a null object reference, which causes an error. For example, if you pass the name of a control on a window that is being closed, that control will not exist when a script accesses the parameter.

See also the usage notes for OpenTab, all of which apply to OpenTabWithParm.

Examples

The following statement opens an instance of a user object *u_data* of type u_benefit_plan as the last tab page in the Tab control tab_1. The parameter "Benefits" is stored in message.StringParm:

```
UserObject u_data
tab_1.OpenTabWithParm(u_data, &
      "Benefits", "u_benefit_plan", 0)
```

These statements open a user object of the type specified in the string *s_u_name* and store the reference to the user object in the variable *u_to_open*. The script gets the value of *s_u_name*, the type of user object to open, from the database. The parameter is the text of the SingleLineEdit sle_loc, so it is stored in Message.StringParm. The user object becomes the third tab page in the Tab control tab_1:

```
UserObject u_to_open
string s_u_name, e_location

e_location = sle_location.Text

SELECT next_userobj INTO  : s_u_name
FROM routing_table
WHERE ...  ;

tab_1.OpenTabWithParm(u_to_open, &
      e_location, s_u_name, 3)
```

The following statements open a user object of the type specified in the string *s_u_name* and store the reference to the user object in the variable *u_to_open*. The parameter is numeric so it is stored in message.DoubleParm. The user object becomes the first tab page in the Tab control tab_1:

```
UserObject u_to_open
integer age = 60
string s_u_name

s_u_name = sle_user.Text
tab_1.OpenTabWithParm(u_to_open, age, &
      s_u_name, 1)
```

See also

OpenTab

OpenUserObject

Adds a user object to the specified window and makes all its properties and controls available to scripts.

To	Use
Open an instance of a particular user object	Syntax 1
Open a user object, allowing the application to select the user object's type during execution	Syntax 2

Syntax 1 For user objects of a known datatype

Description

Opens a user object of a known datatype.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Window objects

Syntax

windowname.**OpenUserObject** (*userobjectvar* {, *x*, *y* })

Argument	Description
windowname	The name of the window in which you want to open the user object.
userobjectvar	The name of the user object you want to display. You can specify a user object defined in the User Object painter (which is a user object datatype) or a variable of the desired user object datatype. OpenUserObject places a reference to the opened user object in *userobjectvar*.
x (optional)	The x coordinate in PowerBuilder units of the user object within the window's frame. The default is 0.
y (optional)	The y coordinate in PowerBuilder units of the user object within the window's frame. The default is 0.

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is null, OpenUserObject returns null.

Usage

Use Syntax 1 when you know what user object you want to open. Use Syntax 2 when the application will determine what type of user object to open when the script runs.

You must open a user object before you can access the properties of the user object. If you access the user object's properties before you open it, an execution error will occur.

A user object that is part of a window's definition (that is, it was added to the window in the Window painter) does not have to opened in a script. PocketBuilder opens it when it opens the window.

OpenUserObject adds the newly opened user object to the window's Control array, which is a property that lists the window's controls.

When you open a user object during execution, the window does not destroy the user object automatically when you close the window. You need to call CloseUserObject to destroy the user object, usually when the window closes. If you do not destroy the user object, it holds on to its allocated memory, resulting in a memory leak.

PocketBuilder displays the user object when it next updates the display or at the end of the script, whichever comes first. For example, if you open several user objects in a script, they will all display at once when the script is complete, unless some other statements cause a change in the screen's appearance (for example, the MessageBox function displays a message or the script changes a visual property of a control).

Calling OpenUserObject twice
If you call Syntax 1 twice to open the same user object, PocketBuilder activates the user object twice; it does not open two instances of the user object.

Examples

This statement displays an instance of a user object named u_Employee in the upper left corner of the window w_emp (coordinates 0,0):

```
w_emp.OpenUserObject(u_Employee)
```

The following statements display an instance of a user object *u_to_open* at 200,100 in the window w_empstatus:

```
u_employee u_to_open
w_empstatus.OpenUserObject(u_to_open, 200, 100)
```

The following statement displays an instance of a user object u_data at location 20,100 in w_info:

```
w_info.OpenUserObject(u_data, 20, 100)
```

See also

OpenUserObjectWithParm

Syntax 2 For user objects of unknown datatype

Description

Opens a user object when the datatype of the user object is not known until the script is executed.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Window objects

Syntax

windowname.**OpenUserObject** (*userobjectvar*, *userobjecttype* {, *x*, *y* })

Argument	Description
windowname	The name of the window in which you want to open the user object.
userobjectvar	A variable of datatype DragObject. OpenUserObject places a reference to the opened user object in *userobjectvar*.
userobjecttype	A string whose value is the name of the user object you want to display. The datatype of *userobjecttype* must be a descendant of *userobjectvar*.
x (optional)	The x coordinate in PowerBuilder units of the user object within the window's frame. The default is 0.
y (optional)	The y coordinate in PowerBuilder units of the user object within the window's frame. The default is 0.

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is null, OpenUserObject returns null.

Usage

Use Syntax 1 when you know what user object you want to open. Use Syntax 2 when the application will determine what type of user object to open when the script runs.

You must open a user object before you can access the properties of the user object. If you access the user object's properties before you open it, an execution error will occur.

A user object that is part of a window's definition (that is, it was added to the window in the Window painter) does not have to be opened in a script. PocketBuilder opens it when it opens the window.

OpenUserObject adds the newly opened user object to the window's Control array, which is a property that lists the window's controls.

When you open a user object during execution, the window does not destroy the user object automatically when you close the window. You need to call CloseUserObject to destroy the user object, usually when the window closes. If you do not destroy the user object, it holds on to its allocated memory, resulting in a memory leak.

PocketBuilder displays the user object when it next updates the display or at the end of the script, whichever comes first. For example, if you open several user objects in a script, they will all display at once when the script is complete, unless some other statements cause a change in the screen's appearance (for example, the MessageBox function displays a message or the script changes a visual property of a control).

The userobjecttype argument
When you use Syntax 2, PocketBuilder opens an instance of a user object of the datatype specified in *userobjecttype* and places a reference to this instance in the variable *userobjectvar*. To refer to the instance in scripts, use *userobjectvar*.

If *userobjecttype* is a descendent user object, you can only refer to properties, events, functions, or structures that are part of the definition of *userobjectvar*. For example, if a user event is declared for *userobjecttype*, you cannot reference it.

The object specified in *userobjecttype* is not automatically included in your executable application. To include it, you must save it in a PKD file (PocketBuilder dynamic library) that you deliver with your application.

Examples

The following example displays a user object of the type specified in the string *s_u_name* and stores the reference to the user object in the variable *u_to_open*. The user object is located at 100,200 in the window w_info:

```
DragObject u_to_open
string s_u_name

s_u_name = sle_user.Text
w_info.OpenUserObject(u_to_open, s_u_name, 100, 200)
```

See also

OpenUserObjectWithParm

OpenUserObjectWithParm

Adds a user object to the specified window and makes all its properties and controls available to scripts, as OpenUserObject does. OpenUserObjectWithParm also stores a parameter in the system's Message object so that it is accessible to the opened object.

To	Use
Open an instance of a particular user object	Syntax 1
Open a user object, allowing the application to select the user object's type during execution	Syntax 2

Syntax 1 For user objects of a known datatype

Description

Opens a user object of a known datatype and stores a parameter in the system's Message object.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Window objects

Syntax

windowname.**OpenUserObjectWithParm** (*userobjectvar*, *parameter* {, *x*, *y* })

Argument	Description
windowname	The name of the window in which you want to open the user object.
userobjectvar	The name of the user object you want to display. You can specify a user object defined in the User Object painter (which is a user object datatype) or a variable of the desired user object datatype. OpenUserObjectWithParm places a reference to the opened user object in *userobjectvar*.
parameter	The parameter you want to store in the Message object when the user object is opened. *Parameter* must have one of these datatypes: • String • Numeric • PowerObject
x (optional)	The x coordinate in PowerBuilder units of the user object within the window's frame. The default is 0.
y (optional)	The y coordinate in PowerBuilder units of the user object within the window's frame. The default is 0.

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is null, OpenUserObjectWithParm returns null.

Usage

The system Message object has three properties for storing data. Depending on the datatype of the parameter specified for OpenUserObjectWithParm, scripts for the opened user object would check one of the following properties:

Message object property	Argument datatype
message.DoubleParm	Numeric
message.PowerObjectParm	PowerObject (PocketBuilder objects, including user-defined structures)
message.StringParm	String

In the opened user object, it is a good idea to access the value passed in the Message object immediately because some other script may use the Message object for another purpose.

Avoiding null object references
When you pass a PowerObject as a parameter, you are passing a reference to the object. The object must exist when you refer to it later or you get a null object reference, which causes an error. For example, if you pass the name of a control on a window that is being closed, that control will not exist when a script accesses the parameter.

See also the usage notes for OpenUserObject, all of which apply to OpenUserObjectWithParm.

Examples

This statement displays an instance of a user object named u_Employee in the window w_emp and stores the string `James Newton` in Message.StringParm. The Constructor event script for the user object uses the string parameter as the text of a StaticText control st_empname in the object. The script that opens the user object has the following statement:

```
w_emp.OpenUserObjectWithParm(u_Employee, "Jim Newton")
```

The user object's Constructor event script has the following statement:

```
st_empname.Text = Message.StringParm
```

The following statements display an instance of a user object *u_to_open* in the window w_emp and store a number in message.DoubleParm:

```
u_employee u_to_open
integer age = 50
w_emp.OpenUserObjectWithParm(u_to_open, age)
```

See also

CloseWithReturn
OpenUserObject
OpenWithParm

Syntax 2 For user objects of unknown datatype

Description

Opens a user object when the datatype of the user object is not known until the script is executed. In addition, OpenUserObjectWithParm stores a parameter in the system's Message object so that it is accessible to the opened object.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Window objects

Syntax

windowname.**OpenUserObjectWithParm** (*userobjectvar*, *parameter*, *userobjecttype* {, *x*, *y* })

Argument	Description
windowname	The name of the window in which you want to open the user object.
userobjectvar	A variable of datatype DragObject. OpenUserObjectWithParm places a reference to the opened user object in *userobjectvar*.
parameter	The parameter you want to store in the Message object when the user object is opened. *Parameter* must have one of these datatypes: • String • Numeric • PowerObject
userobjecttype	A string whose value is the datatype of the user object you want to open. The datatype of *userobjecttype* must be a descendant of *userobjectvar*.
x (optional)	The x coordinate in PowerBuilder units of the user object within the window's frame. The default is 0.
y (optional)	The y coordinate in PowerBuilder units of the user object within the window's frame. The default is 0.

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is null, OpenUserObjectWithParm returns null.

Usage

The system Message object has three properties for storing data. Depending on the datatype of the parameter specified for OpenUserObjectWithParm, scripts for the opened user object would check one of the following properties.

Message object property	Argument datatype
message.DoubleParm	Numeric
message.PowerObjectParm	PowerObject (PocketBuilder objects, including user-defined structures)
message.StringParm	String

In the opened user object, it is a good idea to access the value passed in the Message object immediately because some other script may use the Message object for another purpose.

Avoiding null object references
When you pass a PowerObject as a parameter, you are passing a reference to the object. The object must exist when you refer to it later or you will get a null object reference, which causes an error. For example, if you pass the name of a control on a window that is being closed, that control will not exist when a script accesses the parameter.

See also the usage notes for OpenUserObject, all of which apply to OpenUserObjectWithParm.

Examples

The following statement displays an instance of a user object *u_data* of type u_benefit_plan at location 20,100 in the window w_hresource. The parameter "Benefits" is stored in message.StringParm:

```
DragObject u_data
w_hresource.OpenUserObjectWithParm(u_data, &
      "Benefits", "u_benefit_plan", 20, 100)
```

These statements open a user object of the type specified in the string *s_u_name* and store the reference to the user object in the variable *u_to_open*. The script gets the value of *s_u_name*, the type of user object to open, from the database. The parameter is the text of the SingleLineEdit sle_loc, so it is stored in Message.StringParm. The user object is at the default coordinates 0,0 in the window w_info:

```
DragObject u_to_open
string s_u_name, e_location

e_location = sle_location.Text
```

```
SELECT next_userobj INTO  : s_u_name
FROM routing_table
WHERE ...  ;

w_info.OpenUserObjectWithParm(u_to_open, &
      e_location, s_u_name)
```

The following statements display a user object of the type specified in the string *s_u_name* and store the reference to the user object in the variable *u_to_open*. The parameter is numeric so it is stored in message.DoubleParm. The user object is at the coordinates 100,200 in the window w_emp:

```
userobject u_to_open
integer age = 60
string s_u_name

s_u_name = sle_user.Text
w_emp.OpenUserObjectWithParm(u_to_open, age, &
      s_u_name, 100, 200)
```

See also

CloseWithReturn
OpenUserObject
OpenWithParm

OpenWithParm

Displays a window and makes all its properties and controls available to scripts, as Open does. OpenWithParm also stores a parameter in the system's Message object so that it is accessible to the opened window.

To	Use
Open an instance of a particular window datatype	Syntax 1
Open a window, allowing the application to select the window's datatype during execution	Syntax 2

Syntax 1 For windows of a known datatype

Description

Opens a window object of a known datatype. OpenWithParm displays the window and makes all its properties and controls available to scripts. It also stores a parameter in the system's Message object.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Window objects

Syntax

OpenWithParm (*windowvar*, *parameter* {, *parent* })

Argument	Description
windowvar	The name of the window you want to display. You can specify a window object defined in the Window painter (which is a window datatype) or a variable of the desired window datatype. OpenWithParm places a reference to the open window in *windowvar*.
parameter	The parameter you want to store in the Message object when the window is opened. *Parameter* must have one of these datatypes: • String • Numeric • PowerObject
parent (child and pop-up windows only) (optional)	The window you want make the parent of the child or pop-up window you are opening. If you open a child or pop-up window and omit *parent*, PowerBuilder associates the window being opened with the currently active window.

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is null, OpenWithParm returns null.

Usage

The system Message object has three properties for storing data. Depending on the datatype of the parameter specified for OpenWithParm, your scripts for the opened window would check one of the following properties.

Message object property	Argument datatype
Message.DoubleParm	Numeric
Message.PowerObjectParm	PowerObject (PocketBuilder objects, including user-defined structures)
Message.StringParm	String

In the opened window, it is a good idea to access the value passed in the Message object immediately because some other script may use the Message object for another purpose.

Avoiding null object references When you pass a PowerObject as a parameter, you are passing a reference to the object. The object must exist when you refer to it later or you will get a null object reference, which causes an error. For example, if you pass the name of a control on a window that is being closed, that control will not exist when a script accesses the parameter.

Passing several values as a structure To pass several values, create a user-defined structure to hold the values and access the PowerObjectParm property of the Message object in the opened window. The structure is passed by value, not by reference, so you can access the information even if the original structure has been destroyed.

See also the usage notes for Open, all of which apply to OpenWithParm.

Examples

This statement opens an instance of a window named w_employee and stores the string parameter in Message.StringParm. The script for the window's Open event uses the string parameter as the text of a StaticText control st_empname. The script that opens the window has the following statement:

```
OpenWithParm(w_employee, "James Newton")
```

The window's Open event script has the following statement:

```
st_empname.Text = Message.StringParm
```

The following statements open an instance of a window of the type w_employee. Since the parameter is a number it is stored in Message.DoubleParm:

```
w_employee w_to_open
integer age = 50
OpenWithParm(w_to_open, age)
```

The following statement opens an instance of a child window named cw_data and makes w_employee the parent. The window w_employee must already be open. The parameter *benefit_plan* is a string and is stored in Message.StringParm:

```
OpenWithParm(cw_data, "benefit_plan", w_employee)
```

See also

CloseWithReturn
Open

Syntax 2 For windows of unknown datatype

Description

Opens a window object when you do not know its datatype until the application is running. OpenWithParm displays the window and makes all its properties and controls available to scripts. It also stores a parameter in the system's Message object.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Window objects

Syntax

OpenWithParm (*windowvar*, *parameter*, *windowtype* {, *parent* })

Argument	Description
windowvar	A window variable, usually of datatype window. OpenWithParm places a reference to the open window in *windowvar*.
parameter	The parameter you want to store in the Message object when the window is opened. *Parameter* must have one of these datatypes: • String • Numeric • PowerObject
windowtype	A string whose value is the datatype of the window you want to open. The datatype of *windowtype* must be the same or a descendant of *windowvar*.
parent (child and pop-up windows only)	The window you want to make the parent of the child or pop-up window you are opening. If you open a child or pop-up window and omit *parent*, PowerBuilder associates the window being opened with the currently active window.

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is null, OpenWithParm returns null.

Usage

The system Message object has three properties for storing data. Depending on the datatype of the parameter specified for OpenWithParm, your scripts for the opened window would check one of the following properties.

Message object property	Argument datatype
Message.DoubleParm	Numeric
Message.PowerObjectParm	PowerObject (PocketBuilder objects, including user-defined structures)
Message.StringParm	String

In the opened window, it is a good idea to access the value passed in the Message object immediately because some other script may use the Message object for another purpose.

Avoiding null object references When you pass a PowerObject as a parameter, you are passing a reference to the object. The object must exist when you refer to it later or you will get a null object reference, which causes an error. For example, if you pass the name of a control on a window that is being closed, that control will not exist when a script accesses the parameter.

Passing several values as a structure To pass several values, create a user-defined structure to hold the values and access the PowerObjectParm property of the Message object in the opened window. The structure is passed by value, not by reference, so you can access the information even if the original structure has been destroyed.

See also the usage notes for Open, all of which apply to OpenWithParm.

Examples

These statements open a window of the type specified in the string *s_w_name* and store the reference to the window in the variable *w_to_open*. The script gets the value of *s_w_name*, the type of window to open, from the database. The parameter in *e_location* is text, so it is stored in Message.StringParm:

```
window w_to_open
string s_w_name, e_location

e_location = sle_location.Text

SELECT next_window INTO :s_w_name
FROM routing_table
WHERE ... ;

OpenWithParm(w_to_open, e_location, s_w_name)
```

The following statements open a window of the type specified in the string *c_w_name*, store the reference to the window in the variable *wc_to_open*, and make w_emp the parent window of *wc_to_open*. The parameter is numeric, so it is stored in Message.DoubleParm:

```
window wc_to_open
string c_w_name
integer age = 60

c_w_name = "w_c_emp1"

OpenWithParm(wc_to_open, age, c_w_name, w_emp)
```

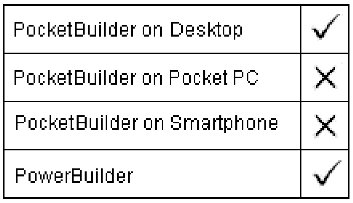
See also

CloseWithReturn
Open

OutgoingCallList

Description

Provides a list of the calls to other routines included in a performance analysis model.



PocketBuilder on Desktop	✓
PocketBuilder on Pocket PC	✗
PocketBuilder on Smartphone	✗
PowerBuilder	✓

Applies to

ProfileLine and ProfileRoutine objects

Syntax

instancename.**OutgoingCallList** (*list*, *aggregate*)

Argument	Description
instancename	Instance name of the ProfileLine or ProfileRoutine object.
list	An unbounded array variable of datatype ProfileCall in which OutgoingCallList stores a ProfileCall object for each call to other routines from within this routine. This argument is passed by reference.
aggregate (ProfileRoutine only)	A boolean indicating whether duplicate routine calls will result in the creation of a single or of multiple ProfileCall objects.

Return value

ErrorReturn. Returns one of the following values:

- Success!—The function succeeded
- ModelNotExistsError!—The model does not exist

Usage

You use the OutgoingCallList function to extract a list of the calls from a line and/or routine to other routines in a performance analysis model. You must have previously created the performance analysis model from a trace file using the BuildModel function. Each caller is defined as a ProfileCall object and provides the called routine and the calling routine, the number of times the call was made, and the elapsed time. The routines are listed in no particular order.

The *aggregate* argument indicates whether duplicate routine calls result in the creation of a single or of multiple ProfileCall objects. This argument has no effect unless line tracing is enabled and a calling routine calls the current routine from more than one line. If *aggregate* is true, a new ProfileCall object is created that aggregates all calls from the calling routine to the current routine. If *aggregate* is false, multiple ProfileCall objects are returned, one for each line from which the calling routine called the called routine.

Examples

This example gets a list of the routines included in a performance analysis model and then gets a list of the routines called by each routine:

```
Long ll_cnt
ProfileCall lproc_call[]

lpro_model.BuildModel()
lpro_model.RoutineList(iprort_list)

FOR ll_cnt = 1 TO UpperBound(iprort_list)

iprort_list[ll_cnt].OutgoingCallList(lproc_call, &
          TRUE)
      ...
NEXT
```

See also

BuildModel
IncomingCallList

PageCount

Description Returns the total number of pages in the document in a RichTextEdit control.

PocketBuilder	✗
PowerBuilder	✓

Applies to RichTextEdit controls

Syntax *rtename*.**PageCount** ( )

Return value Integer. Returns the number of pages in the RichTextEdit control. Returns 1 if the control contains no text and -1 if an error occurs.

PageCreated

Description Reports whether a tab page has been created.

PocketBuilder	✗
PowerBuilder	✓

Applies to User objects used as tab pages

Syntax *userobject*.**PageCreated** ( )

Return value Boolean. Returns true if the user object is a tab page and has been created and false if the user object is not a tab page or has not been created.

ParentWindow

Description Obtains the parent window of a window.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to Window objects

Syntax

windowname.**ParentWindow** ( )

Argument	Description
windowname	The name of a window for which you want to obtain the parent object

Return value

Window. Returns the parent of *windowname*. Returns a null object reference if an error occurs or if *windowname* is null.

Usage

The ParentWindow function, along with the pronoun Parent, allows you to write more general scripts by avoiding the coding of actual window names. Parent refers to the window that contains the current object or control—the local environment. ParentWindow returns the parent window of a specified window.

Whether a window has a parent depends on its type and how it was opened. You can specify the parent when you open the window. For windows that always have parents, PocketBuilder chooses the parent if you do not specify it. Response windows always have a parent window.

The ParentWindow property of the Menu object can be used like a pronoun in Menu scripts. It identifies the window with which the menu is associated when your program is running. For more information, see the *Users Guide*.

Examples

These statements return the parent of child_1. The parent is a window of the datatype Win1:

```
Win1 w_parent
w_parent = child_1.ParentWindow()
```

The following script for a Cancel button in a pop-up window triggers an event for the parent window of the button's parent window (the window that contains the button). Then it closes the button's window. The parent window of that window will have a script for the cancelrequested event:

```
Parent.ParentWindow().TriggerEvent("cancelrequested")
Close(Parent)
```

Paste

Description

Inserts (pastes) the contents of the clipboard into the specified control. For editable controls, text on the clipboard is pasted at the insertion point. For OLE controls, the OLE object on the clipboard replaces any object already in the control.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

EditMask, MultiLineEdit, SingleLineEdit, RichTextEdit, DropDownListBox, DropDownPictureListBox, DataWindow, OLE controls

Syntax

controlname.**Paste** ( )

Argument	Description
controlname	The name of the control into which you want to insert the contents of the clipboard. If *controlname* is a DataWindow, text is pasted into the edit control over the current row and column. If *controlname* is a DropDownListBox the AllowEdit property must be true

Return value

Long. If *controlname* is null, Paste returns null.

For edit controls, returns the number of characters that were pasted into *controlname*. If nothing has been cut or copied (the clipboard is empty), the Paste function does not change the contents of the edit control and returns 0. If the clipboard contains nontext data (for example, a bitmap or OLE object) and the control cannot accept that data, Paste does not change the contents and returns 0.

Usage

For editable controls, if text is selected in *controlname*, Paste replaces the text with the contents of the clipboard. If the clipboard contains more lines than fit in the edit control, only the number of lines that fit are pasted.

In a DataWindow control, the text is pasted into the edit control over the current row and column. If the clipboard contains more text that is allowed for that column, the text is truncated. If the clipboard text does not match the column's datatype, all the text is truncated, so that any selected text is replaced with an empty string.

To insert a specific string in *controlname* or to replace selected text with a specific string, use the ReplaceText function.

Examples

If the clipboard contains `Proposal good for 90 days` and no text is selected, this statement pastes `Proposal good for 90 days` in mle_Comment1 at the insertion point and returns 25:

```
mle_Comment1.Paste()
```

If the clipboard contains the string `Final Edition`, mle_Comment2 contains `This is a Preliminary Draft`, and the text in mle_Comment2 is selected, this statement deletes `This is a Preliminary Draft`, replaces it with `Final Edition`, and returns 13:

```
mle_Comment2.Paste()
```

See also

Copy
Cut
PasteLink
PasteSpecial
ReplaceText

PasteLink

Description

Pastes a link to the contents of the clipboard into the control. The server application for the object on the clipboard must be running.

PocketBuilder	✗
PowerBuilder	✓

Applies to

OLE controls

Syntax

olecontrol.**PasteLink** ( )

Return value

Integer. Returns 0 if it succeeds and a negative number if an error occurs.

PasteRTF

Description

Pastes rich text data from a string into a DataWindow control, DataStore object, or RichTextEdit control.

PocketBuilder	✗
PowerBuilder	✓

Applies to DataWindow controls, DataStore objects, and RichTextEdit controls

Syntax *rtename*.**PasteRTF** (*richtextstring*, { *band* })

Return value Long. Returns the number of characters pasted if it succeeds and -1 if an error occurs. If *richtextstring* is null, PasteRTF returns null.

PasteSpecial

Description Displays a standard OLE dialog allowing the user to choose whether to embed or link the OLE object on the clipboard when pasting it in the specified control. Embedding is the equivalent of calling the Paste function, and linking is the same as calling PasteLink.

PocketBuilder	✗
PowerBuilder	✓

Applies to OLE controls

Syntax *olecontrol*.**PasteSpecial** ( )

Argument	Description
olecontrol	The name of the OLE control into which you want to paste the object on the clipboard

Return value Integer. Returns 0 if it succeeds and one of the following values if an error occurs:

1	User canceled without selecting a paste option
-1	No data found
-9	Other error

If *ole2control* is null, PasteSpecial returns null.

Pi

Description Multiplies pi by a specified number.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

Pi (*n*)

Argument	Description
n	The number you want to multiply by pi (3.14159265358979323...)

Return value

Double. Returns the result of multiplying *n* by pi if it succeeds and -1 if an error occurs. If *n* is null, Pi returns null.

Usage

Use Pi to convert angles to and from radians.

Examples

This statement returns pi:

```
Pi(1)
```

Both these statements return the area of a circle with the radius *id_Rad*, an instance variable of type double:

```
Pi(1) * id_Rad^2

Pi(id_Rad^2)
```

The following statements compute the cosine of a 45-degree angle:

```
real degree = 45.0, cosine
cosine = Cos(degree * (Pi(2)/360))
```

See also

Cos
Sin
Tan
Pi method for DataWindows in the *DataWindow Reference*

PixelsToUnits

Description

Converts pixels to PowerBuilder units. Because pixels are not usually square, you also specify whether you are converting the pixels' horizontal or vertical measurement.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

PixelsToUnits (*pixels*, *type*)

Argument	Description
pixels	An integer whose value is the number of pixels you want to convert to PowerBuilder units.
type	A value of the ConvertType enumerated datatype value indicating how to convert the value: • XPixelsToUnits! — Convert the pixels in the horizontal direction. • YPixelsToUnits! — Convert the pixels in the vertical direction.

Return value

Integer. Returns the converted value if it succeeds and -1 if an error occurs. If any argument's value is null, PixelsToUnits returns null.

Examples

These statements convert 35 horizontal pixels to PowerBuilder units and set the variable *Value* equal to the converted value:

```
integer Value
Value = PixelsToUnits(35, XPixelsToUnits!)
```

See also

UnitsToPixels

PointerX

Description

Determines the distance of the pointer from the left edge of the specified object.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Any object or control

Syntax

objectname.**PointerX** ( )

Argument	Description
objectname	The name of the control or window for which you want the pointer's distance from the left edge. If you do not specify *objectname*, PointerX reports the distance from the left edge of the current sheet or window.

Return value

Integer. Returns the pointer's distance from the left edge of *objectname* in PowerBuilder units if it succeeds and -1 if an error occurs. If *objectname* is null, PointerX returns null.

Examples

In a script for a control in a window, the following example stores the distance of the pointer from the edge of the window in the variable *li_dist*. If the pointer is 5 units from the left edge of the window, *li_dist* equals 5:

```
integer li_dist
li_dist = Parent.PointerX()
```

This statement in a control's RButtonDown script displays a pop-up menu m_Appl.M_Help at the cursor position:

```
m_Appl.m_Help.PopMenu(Parent.PointerX(), &
   Parent.PointerY())
```

If the previous example was part of the window's RButtonDown script, instead of a control in the window, the following statement displays the pop-up menu at the cursor position:

```
m_Appl.m_Help.PopMenu(This.PointerX(), &
   This.PointerY())
```

See also

PointerY
PopMenu
WorkSpaceHeight
WorkSpaceWidth
WorkSpaceX
WorkSpaceY

PointerY

Description

Determines the distance of the pointer from the top of the specified object.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Any object or control

Syntax

objectname.**PointerY** ( )

Argument	Description
objectname	The name of the control or window for which you want the pointer's distance from the top. If you do not specify *objectname*, PointerY reports the distance from the top of the current sheet or window.

Return value

Integer. Returns the pointer's distance from the top of *objectname* in PowerBuilder units if it succeeds and -1 if an error occurs. If *objectname* is null, PointerY returns null.

Examples

In a script for a control in a window, the following example stores the distance of the pointer from the top of the window in the variable *li_dist*. If the pointer is 10 units from the top of the window, *li_dist* equals 10:

```
integer li_Dist
li_Dist = Parent.PointerY()
```

This statement in a control's RButtonDown script displays a pop-up menu m_Appl.M_Help at the cursor position:

```
m_Appl.M_Help.PopMenu(Parent.PointerX(), &
   Parent.PointerY())
```

See also

PointerX
PopMenu
WorkSpaceHeight
WorkSpaceWidth
WorkSpaceX
WorkSpaceY

PopMenu

Description

Displays a menu at the specified location.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Menu objects

Syntax

menuname.**PopMenu** (*xlocation*, *ylocation*)

Argument	Description
menuname	The fully qualified name of a menu on a menu bar you want to display at the specified location
xlocation	The distance in PowerBuilder units of the displayed menu from the left edge of the window
ylocation	The distance in PowerBuilder units of the displayed menu from the top of the window

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is null, PopMenu returns null.

Usage

If the menu object is not associated with the window so that it was opened when the window was opened, you must use CREATE to allocated memory for the menu (see the last example).

If the Visible property of the menu is false, you must make the menu visible before you can display it as a pop-up menu.

The coordinates you specify for PopMenu are relative to the active window. In an MDI application, the coordinates are relative to the frame window, which is the active window. To display a menu at the cursor position, call PointerX and PointerY for the active window (the frame window in an MDI application) to get the coordinates of the cursor. (See the examples.)

Calling *PopMenu* in an object script
PopMenu must be called in an object script. It should not be called in a global function.

Examples

These statements display the menu m_Emp.M_Procedures at location 100, 200 in the active window. M_Emp is the menu associated with the window:

```
m_Emp.M_Procedures.PopMenu(100, 200)
```

This statement displays the menu m_Appl.M_File at the cursor position, where m_Appl is the menu associated with the window.

```
m_Appl.M_file.PopMenu(PointerX(), PointerY())
```

These statements display a pop-up menu at the cursor position. Menu4 was created in the Menu painter and includes a menu called m_language. Menu4 is not the menu for the active window. *NewMenu* is an instance of Menu4 (datatype Menu4):

```
Menu4 NewMenu
NewMenu = CREATE Menu4
NewMenu.m_language.PopMenu(PointerX(), PointerY())
```

PopulateError

Description

Fills in the Error object without causing a SystemError event.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

PopulateError (*number*, *text*)

Argument	Description
number	The integer to be stored in the number property of the Error object
text	The string to be stored in text property of the Error object

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs. The return value is usually not used.

Usage

If the values you want to populate the Error object with depend on the current value of a variable in your script, you can use PopulateError to assign values to the number and text fields in the Error object (the remaining fields of the Error object will be populated automatically, including the line number of the error). Then you can call SignalError without arguments to trigger a SystemError. You will need to include code in the SystemError event script to recognize and handle the error you have created. If there is no script for the SystemError event, the SignalError function does nothing.

Examples

The gf_DoSomething function takes a table name and a record and returns 0 for success and a negative number for an error. The following statements set the number and text values in the Error object according to a script variable, then trigger a SystemError event once the processing is complete:

```
li_result = gf_DoSomething("Company", record_id)
```

```
IF (li_result < 0) THEN
   CHOOSE CASE li_result
   CASE -1
      PopulateError(1, "No company record exists &
      record id: " + record_id)
   CASE -2
      PopulateError(2, "That company record is &
      currently locked. Please try again later.")
   CASE -3
      PopulateError(3, "The company record could &
      not be updated.")
   CASE else
      PopulateError(999, "Update failed.")
   END CHOOSE
   SignalError()
END IF
```

See also

SignalError

Pos

Description

Finds one string within another string.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

Pos (*string1*, *string2* {, *start* })

Argument	Description
string1	The string in which you want to find *string2*.
string2	The string you want to find in *string1*.
start (optional)	A long indicating where the search will begin in *string1*. The default is 1.

Return value

Long. Returns a long whose value is the starting position of the first occurrence of *string2* in *string1 after* the position specified in *start*. If *string2* is not found in *string1* or if *start* is not within *string1*, Pos returns 0. If any argument's value is null, Pos returns null.

Usage

The Pos function is case sensitive.

Examples

This statement returns 6:

```
Pos("BABE RUTH", "RU")
```

This statement returns 1:

```
Pos("BABE RUTH", "B")
```

This statement returns 0, because the case does not match:

```
Pos("BABE RUTH", "be")
```

This statement starts searching at position 4 and returns 0, because position 4 is after the occurrence of BE:

```
Pos("BABE RUTH", "BE", 4 )
```

These statements change the text NY in the SingleLineEdit sle_group to North East:

```
long place_nbr
place_nbr = Pos(sle_group.Text, "NY")
sle_group.SelectText(place_nbr, 2)
sle_group.ReplaceText("North East")
```

These statements separate the return value of GetBandAtPointer into the band name and row number. The Pos function finds the position of the tab in the string and the Left and Mid functions extract the information to the left and right of the tab:

```
string s, ls_left, ls_right
integer li_tab

s = dw_groups.GetBandAtPointer()
li_tab = Pos(s, "~t", 1)

ls_left = Left(s, li_tab - 1)
ls_right = Mid(s, li_tab + 1)
```

You could write similar code for a generic parsing function with three arguments. The string *s* would be an argument passed by value and *ls_left* and *ls_right* would be strings passed by reference.

Other functions that return a pair of tab-separated values for which you could use the parsing function are GetObjectAtPointer and GetValue.

See also

GetValue method for DataWindows in the *DataWindow Reference*
GetObjectAtPointer method for DataWindows in the *DataWindow Reference*
LastPos
Left
Mid
Right
Pos method for DataWindows in the *DataWindow Reference*

PosW

Description

Finds one string within another string.

PocketBuilder	✗
PowerBuilder	✓

Obsolete function
PosW is an obsolete function. It has the same behavior as Pos.

Syntax

PosW (*string1*, *string2* {, *start* })

Return value

Long. Returns a long whose value is the starting position of the first occurrence of *string2* in *string1* *after* the position specified in *start*.

Position

Reports the position of the insertion point in an editable control.

To report	Use
The position of the insertion point in any editable control (except RichTextEdit)	Syntax 1
The position of the insertion point or the start and end of selected text in a RichTextEdit control or a DataWindow whose object has the RichTextEdit presentation style	Syntax 2

Syntax 1 For editable controls, except RichTextEdit

Description

Determines the position of the insertion point in an edit control.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

DataWindow, EditMask, MultiLineEdit, SingleLineEdit, or DropDownListBox, DropDownPictureListBox controls

Syntax

editname.**Position** ( )

Argument	Description
editname	The name of the control in which you want to find the location of the insertion point

Return value

Long. Returns the location of the insertion point in *editname* if it succeeds and -1 if an error occurs. If *editname* is null, Position returns null.

Usage

Position reports the position number of the character immediately following the insertion point. For example, Position returns 1 if the cursor is at the beginning of *editname*. If text is selected in *editname*, Position reports the number of the first character of the selected text.

In a DataWindow control, Position reports the insertion point's position in the edit control over the current row and column.

Examples

If mle_EmpAddress contains Boston Street, the cursor is immediately after the n in Boston, and no text is selected, this statement returns 7:

```
mle_EmpAddress.Position()
```

If mle_EmpAddress contains Boston Street and Street is selected, this statement returns 8 (the position of the S in Street):

```
mle_EmpAddress.Position()
```

See also

SelectedLine
SelectedStart

Syntax 2 For RichTextEdit controls

Description

Determines the line and column position of the insertion point or the start and end of selected text in an RichTextEdit control.

PocketBuilder	✗
PowerBuilder	✓

Applies to

RichTextEdit and DataWindow controls

Syntax

rtename.**Position** (*fromline*, *fromchar* {, *toline*, *tochar* })

Return value

Band enumerated datatype. Returns the band (Detail!, Header!, or Footer!) containing the selection or insertion point.

Post

Description

Adds a message to the message queue for a window, either a PocketBuilder window or window of another application.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

Post (*handle*, *message#*, *word*, *long*)

Argument	Description
handle	A long whose value is the system handle of a window (that you have created in PocketBuilder or another application) to which you want to post a message.
message#	An UnsignedInteger whose value is the system message number of the message you want to post.
word	A long whose value is the integer value of the message. If this argument is not used by the message, enter 0.
long	The long value of the message or a string.

Return value

Boolean. If any argument's value is null, Post returns null.

Usage

Use Post or Send when you want to trigger system events that are not PocketBuilder-defined events. Post is asynchronous; it adds a message to the end of the window's message queue. Send is synchronous; its message triggers an event immediately.

To obtain the handle of a PocketBuilder window, use the Handle function.

To trigger PocketBuilder events, use TriggerEvent or PostEvent. These functions run the script associated with the event. They are easier to code and bypass the messaging queue.

When you specify a string for *long*, Post stores a copy of the string and passes a pointer to it.

Examples

This statement scrolls the window w_date down one page after all the previous messages in the message queue for the window have been processed:

```
Post(Handle(w_date), 277, 3, 0)
```

See also

Handle
PostEvent
Send
TriggerEvent

PostEvent

Description

Adds an event to the end of the event queue of an object.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Any object, except the application object

Syntax

objectname.**PostEvent** (*event*, { *word*, *long* })

Argument	Description
objectname	The name of any PocketBuilder object or control (except an application) that has events associated with it.
event	A value of the TrigEvent enumerated datatype that identifies a PocketBuilder event (for example, Clicked!, Modified!, or DoubleClicked!) or a string whose value is the name of an event. The event must be a valid event for *objectname* and a script must exist for the event in *objectname*.

Argument	Description
word (optional)	A long value to be stored in the WordParm property of the system's Message object. If you want to specify a value for *long*, but not *word*, enter 0. (For cross-platform compatibility, WordParm and LongParm are both longs).
long (optional)	A long value or a string that you want to store in the LongParm property of the system's Message object. When you specify a string, a pointer to the string is stored in the LongParm property, which you can access with the String function (see Usage).

Return value

Boolean. Returns true if it is successful and false if the event is not a valid event for *objectname* or no script exists for the event in *objectname*. If any argument's value is null, PostEvent returns null.

Usage

You cannot post events to the event queue for an application object. Use TriggerEvent instead.

You cannot post or trigger events for objects that do not have events, such as drawing objects. You cannot post or trigger events in a batch application that has no user interface because the application has no event queue.

After you call PostEvent, check the return code to determine whether PostEvent succeeded.

You can pass information to the event script with the *word* and *long* arguments. The information is stored in the Message object. In your script, you can reference the WordParm and LongParm fields of the Message object to access the information. Note that the Message object is saved and restored just before the posted event script runs so that the information you passed is available even if other code has used the Message object too.

If you have specified a string for *long*, you can access it in the triggered event by using the String function with the keyword "address" as the *format* parameter. (Note that PocketBuilder has stored the string at an arbitrary memory location and you are relying on nothing else having altered the pointer or the stored string.) Your event script might begin as follows:

```
string PassedString
PassedString = String(Message.LongParm, "address")
```

TriggerEvent and PostEvent are useful for preventing duplication of code. If two controls perform the same task, you can use PostEvent in one control's event script to execute the other's script, instead of repeating the code in two places. For example, if both a button and a menu delete data, the button's Clicked script can perform the deletion and the menu's Clicked event script can post an event that runs the button's Clicked event script.

Choosing PostEvent or TriggerEvent Both PostEvent and TriggerEvent cause event scripts to be executed. PostEvent is asynchronous; it adds the event to the end of an object's event queue. TriggerEvent is synchronous; the event is triggered immediately.

Use PostEvent when you want the current event script to complete before the posted event script runs. TriggerEvent interrupts the current script to run the triggered event's script. Use it when you need to interrupt a process, such as canceling printing.

If the function is the last line in an event script and there are no other events pending, PostEvent and TriggerEvent have the same effect.

Events and messages in Windows Both PostEvent and TriggerEvent cause a script associated with an event to be executed. However, these functions do not send the actual event message. This is important when you are choosing the target object and event. The following background information explains this concept.

Many PocketBuilder functions send Windows messages, which in turn trigger events and run scripts. For example, the Close function sends a Windows close message (WM_CLOSE). PocketBuilder maps the message to its internal close message (PBM_CLOSE), then runs the Close event's script and closes the window.

If you use TriggerEvent or PostEvent with Close! as the argument, PocketBuilder runs the Close event's script but it does *not* close the window because it did not receive the close message. Therefore, the choice of which event to trigger is important. If you trigger the Clicked! event for a button whose script calls the Close function, PocketBuilder runs the Close event's script *and* closes the window.

Use Post or Send when you want to trigger system events that are not PocketBuilder-defined events.

Examples

This statement adds the Clicked event to the event queue for CommandButton cb_OK. The event script will be executed after any other pending event scripts are run:

```
cb_OK.PostEvent(Clicked!)
```

This statement adds the user-defined event cb_exit_request to the event queue in the parent window:

```
Parent.PostEvent("cb_exit_request")
```

This example posts an event for cb_exit_request with an argument and then retrieves that value from the Message object in the event's script.

The first part of the example is code for a button in a window. It adds the user-defined event cb_exit_request to the event queue in the parent window. The value 455 is stored in the Message object for the use of the event's script:

```
Parent.PostEvent("cb_exit_request", 455, 0)
```

The second part of the example is the beginning of the cb_exit_request event script, which assigns the value passed in the Message object to a local variable. The script can use the value in whatever way is appropriate to the situation:

```
integer numarg
numarg = Message.WordParm
```

See also

Post
Send
TriggerEvent

PostURL

Description

Performs an HTTP Post, allowing a PowerBuilder application to send a request through CGI, NSAPI, or ISAPI.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Inet objects

Syntax

servicereference.**PostURL** (*urlname*, *urldata*, *headers*, {*serverport*, } *data*)

Argument	Description
servicereference	Reference to the Internet service instance.
urlname	String specifying the URL to post.
urldata	Blob specifying arguments to the URL specified by *urlname*.
headers	String specifying HTML headers. In Netscape, a newline (~n) is required after each HTTP header and a final newline after all headers.
serverport (optional)	Specifies the server port number for the request. The default value for this argument is 0, which means that the port number is determined by the system (port 80 for HTTP requests).
data	InternetResult instance into which the function returns HTML.

Return value

Integer. Returns one of the following values:

1	Success
-1	General error
-2	Invalid URL
-4	Cannot connect to the Internet
-5	Unsupported secure (HTTPS) connection attempted
-6	Internet request failed

Usage

Call this function to invoke a CGI, NSAPI, or ISAPI function.

Data references a standard class user object that descends from InternetResult and that has an overridden InternetData function. This overridden function then performs the required processing with the returned HTML. Because the Internet returns data asynchronously, *data* must reference a variable that remains in scope after the function executes (such as a window-level instance variable).

To simulate a form submission, you need to send a header that indicates the proper Content-Type. For forms, the proper Content-Type header is:

Content-Type: application/x-www-form-urlencoded

Examples

This example calls the PostURL function using server port 8080. *Iinet* is an instance variable of type inet:

```
Blob lblb_args
String ls_headers
String ls_url
Long ll_length
```

```
iir_msgbox = CREATE n_ir_msgbox
ls_url = "http://coltrane.sybase.com/"
ls_url += "cgi-bin/pbcgi60.exe/"
ls_url += "myapp/n_cst_html/f_test?"
lblb_args = blob("")
ll_length = Len(lblb_args)
ls_headers = "Content-Length: " &
   + String(ll_length) + "~n~n"
iinet.PostURL &
   (ls_url, lblb_args, ls_headers, 8080, iir_msgbox)
```

This example shows the use of a header with the correct content-type for a form:

```
Blob lblb_args
String ls_headers
String ls_url
String ls_args
long ll_length
integer li_rc

li_rc = GetContextService( "Internet", iinet_base )
IF li_rc = 1 THEN
   ir = CREATE n_ir
   ls_url = "http://localhost/Site/testurl.stm?"
   ls_args = "user=MyName&pwd=MyPasswd"
   lblb_args = Blob( ls_args )
   ll_length = Len( lblb_args )
   ls_header = "Content-Type: " + &
      "application/x-www-form-urlencoded~n" + &
      "Content-Length: " + String( ll_length ) + "~n~n"
   li_rc = iinet.PostURL( ls_url, lblb_args, &
      ls_header, ir )
END IF
```

See also

GetURL
HyperLinkToURL
InternetData

Preview

Description

Displays the contents of a RichTextEdit control as either a preview of the document as it would print or in an editing view.

PocketBuilder	✗
PowerBuilder	✓

Applies to

RichTextEdit controls

Syntax

rtename.**Preview** (*previewsetting*)

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs.

Print

Sends data to the current printer (or spooler, if the user has a spooler set up). There are several syntaxes.

Required third-party software
You must install the FieldSoftware PrinterCE SDK before you can use print methods in PocketBuilder applications deployed to a device or emulator. An evaluation version of this software is available from the FieldSoftware Web site at http://www.fieldsoftware.com.

For syntax for DataWindows or DataStores, see the Print method for DataWindows in the *DataWindow Reference* or the online Help.

To	Use
Include a visual object, such as a window or a graph control in a print job	Syntax 1
Send one or more lines of text as part of a print job	Syntax 2
Print the contents of an RTE control	Syntax 3

Syntax 1 For printing a visual object in a print job

Description

Includes a visual object, such as a window or a graph control, in a print job that you have started with the PrintOpen function.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✓

Applies to

Any object

Syntax

objectname.**Print** (*printjobnumber*, *x*, *y* {, *width*, *height* })

Argument	Description
objectname	The name of the object that you want to print. The object must either be a window or an object whose ancestor type is DragObject, which includes all the controls that you can place in a window.
printjobnumber	The number the PrintOpen function assigns to the print job.
x	An integer whose value is the x coordinate on the page of the left corner of the object, in thousandths of an inch.
y	An integer whose value is the y coordinate on the page of the left corner of the object, in thousandths of an inch.
width (optional)	An integer specifying the printed width of the object in thousandths of an inch. If omitted, PocketBuilder uses the object's original width.
height (optional)	An integer specifying the printed height of the object in thousandths of an inch. If omitted, PocketBuilder uses the object's original height.

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is null, Print returns null.

Usage

PocketBuilder manages print jobs by opening the job, sending data, and closing the job. When you use Syntax 2 or 3, you must call the PrintOpen and PrintClose functions yourself to manage the process.

Print area and margins The print area is the physical page size minus any margins in the printer itself.

Examples

This example prints the CommandButton cb_close in its original size at location 500, 1000:

```
long Job
Job = PrintOpen( )
cb_close.Print(Job, 500,1000)
PrintClose(Job)
```

This example opens a print job, which defines a new page, then prints a title using the third syntax of Print. Then it uses this syntax of Print to print a graph on the first page and a window on the second page:

```
long Job
Job = PrintOpen( )
Print(Job, "Report of Year-to-Date Sales")
gr_sales1.Print(Job, 1000,PrintY(Job)+500, &
   6000,4500)
PrintPage(Job)
w_sales.Print(Job, 1000,500, 6000,4500)
PrintClose(Job)
```

See also

PrintCancel
PrintClose
PrintOpen
PrintScreen

Syntax 2 For printing text in a print job

Description

Sends one or more lines of text as part of a print job that you have opened with the PrintOpen function. You can specify tab settings before or after the text. The tab settings control the text's horizontal position on the page.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✓

Applies to

Not object-specific

Syntax

Print (*printjobnumber*, { *tab1*, } *string* {, *tab2* })

Argument	Description
printjobnumber	The number the PrintOpen function assigned to the print job.

Argument	Description
tab1 (optional)	The position, measured from the left edge of the print area in thousandths of a inch, to which the print cursor should move before *string* is printed. If the print cursor is already at or beyond the position or if you omit *tab1*, Print starts printing at the current position of the print cursor.
string	The string you want to print. If the string includes carriage return-newline character pairs (~r~n), the string will print on multiple lines. However, the initial tab position is ignored on subsequent lines.
tab2 (optional)	The new position, measured from the left edge of the print area in thousandths of a inch, of the print cursor after *string* printed. If the print cursor is already at or beyond the specified position, Print ignores *tab2* and the print cursor remains at the end of the text. If you omit *tab2*, Print moves the print cursor to the beginning of a new line.

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is null, Print returns null.

Usage

PocketBuilder manages print jobs by opening the job, sending data, and closing the job. When you use Syntax 2 or 3, you must call the PrintOpen and PrintClose functions yourself to manage the process.

Print cursor In a print job, PocketBuilder uses a print cursor to keep track of the print location. The print cursor stores the coordinates of the upper-left corner of the location at which print will being. PocketBuilder updates the print cursor after printing text with Print.

Line spacing when printing text Line spacing in PocketBuilder is proportional to character height. The default line spacing is 1.2 times the character height. When Print starts a new line, it sets the x coordinate of the cursor to 0 and increases the y coordinate by the current line spacing. You can change the line spacing with the PrintSetSpacing function, which lets you specify a new factor to be multiplied by the character height.

Because Syntax 3 of Print increments the y coordinate each time it creates a new line, it also handles page breaks automatically. When the y coordinate exceeds the page size, PocketBuilder automatically creates a new page in the print job. You do not need to call the PrintPage function, as you would if you were using the printing functions that control the cursor position (for example, PrintText or PrintLine).

Print area and margins The print area is the physical page size minus any margins in the printer itself.

Using fonts You can use PrintDefineFont and PrintSetFont to specify the font used by the Print function when you are printing a string.

Fonts for multiple languages The default font for print functions is the system font, but multiple languages cannot be printed correctly using the system font. The Tahoma font typically produces good results. However, if the printer font is set to Tahoma and the Tahoma font is not installed on the printer, PowerBuilder downloads the entire font set to the printer when it encounters a multilanguage character. Use the PrintDefineFont and PrintSetFont functions to specify a font that is available on users' printers and supports multiple languages.

Examples

This example opens a print job, prints the string `Sybase Corporation` in the default font, and then starts a new line:

```
long Job

// Define a blank page and assign the job an ID
Job = PrintOpen( )

// Print the string and then start a new line
Print(Job, "Sybase Corporation")
...
PrintClose(Job)
```

This example opens a print job, prints the string `Sybase Corporation` in the default font, tabs 5 inches from the left edge of the print area but does not start a new line:

```
long Job

// Define a blank page and assign the job an ID
Job = PrintOpen( )

// Print the string but do not start a new line
Print(Job, "Sybase Corporation", 5000)
...
PrintClose(Job)
```

The first Print statement below tabs half an inch from the left edge of the print area, prints the string `Sybase Corporation`, and then starts a new line. The second Print statement tabs one inch from the left edge of the print area, prints the string `Directors:`, and then starts a new line:

```
long Job

// Define a blank page and assign the job an ID
Job = PrintOpen( )
```

```
// Print the string and start a new line
Print(Job, 500, "Sybase Corporation")

// Tab 1 inch from the left edge and print
Print(Job, 1000, "Directors:")
...
PrintClose(Job)
```

The first Print statement below tabs half an inch from the left edge of the print area prints the string `Sybase Corporation`, and then tabs 6 inches from the left edge of the print area but does not start a new line. The second Print statement prints the current date and then starts a new line:

```
long Job

// Define a blank page and assign the job an ID
Job = PrintOpen( )

// Print string and tab 6 inches from the left edge
Print(Job, 500, "Sybase Corporation", 6000)

// Print the current date on the same line
Print(Job, String(Today()))
...
PrintClose(Job)
```

In a window that displays a database error message in a MultiLineEdit mle_message, the following script for a Print button prints a title with the date and time and the message:

```
long li_prt

li_prt = PrintOpen("Database Error")

Print(li_prt, "Database error - " &
   + String(Today(), "mm/dd/yyyy") &
      + " - " &
      + String(Now(), "HH:MM:SS"))
Print(li_prt, " ")
Print(li_prt, mle_message.text)

PrintClose(li_prt)
```

See also

PrintCancel
PrintClose
PrintDataWindow
PrintOpen

PrintScreen
PrintSetFont
PrintSetSpacing

Syntax 3 For RichTextEdit controls

Description

Prints the contents of a RichTextEdit control.

PocketBuilder	✗
PowerBuilder	✓

Applies to

RichTextEdit controls

Syntax

rtename.**Print** (*copies*, *pagerange*, *collate*, *canceldialog*)

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs.

PrintBitmap

Description

Writes a bitmap at the specified location on the current page.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✓

Syntax

PrintBitmap (*printjobnumber*, *bitmap*, *x*, *y*, *width*, *height*)

Argument	Description
printjobnumber	The number the PrintOpen function assigned to the print job.
bitmap	A string whose value is the file name of the bitmap image.
x	An integer whose value is the x coordinate (in thousandths of an inch) on the page of the bitmap image.
y	An integer whose value is the y coordinate (in thousandths of an inch) on the page of the bitmap image.
width	The integer width of the bitmap image in thousandths of an inch. If *width* is 0, PocketBuilder uses the original width of the image.
height	The integer height of the bitmap image in thousandths of an inch. If *height* is 0, PocketBuilder uses the original height of the image.

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is null, PrintBitmap returns null.

Usage

PrintBitmap does not change the position of the print cursor, which remains where it was before the function was called. In general, print functions in which you specify coordinates do not affect the print cursor (see the functions listed in See also).

Required third-party software
You must install the FieldSoftware PrinterCE SDK before you can use print methods in PocketBuilder applications deployed to a Pocket PC device or emulator. An evaluation version of this software is available from the FieldSoftware Web site at http://www.fieldsoftware.com.

Examples

These statements define a new blank page and then print the bitmap in file *d:\PB\BITMAP1.BMP* in its original size at location 50,100:

```
long Job

// Define a new blank page.
Job = PrintOpen( )

// Print the bitmap in its original size.
PrintBitmap(Job, "d:\PB\BITMAP1.BMP", 50,100, 0,0)
// Send the page to the printer and close Job.
PrintClose(Job)
```

See also

PrintClose
PrintLine
PrintRect
PrintRoundRect
PrintOval
PrintOpen

PrintCancel

Description

Cancels printing and deletes the spool file, if any. Cancels printing of a print job that you opened with the PrintOpen function. The print job is identified by the number returned by PrintOpen.

PocketBuilder	✗
PowerBuilder	✓

Syntax **PrintCancel** (*printjobnumber*)

Return value Integer. Returns 1 if it succeeds and -1 if an error occurs. If *printjobnumber* is null, PrintCancel returns null.

PrintClose

Description Sends the current page to the printer (or spooler) and closes the job. Call PrintClose as the last command of a print job unless PrintCancel function has closed the job.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✓

Syntax **PrintClose** (*printjobnumber*)

Argument	Description
printjobnumber	The number the PrintOpen function assigned to the print job

Return value Integer. Returns 1 if it succeeds and -1 if an error occurs. If *printjobnumber* is null, PrintClose returns null.

Usage When you open a print job, you must close (or cancel) it. To avoid hung print jobs, process and close a print job in the same event in which you open it.

Examples This example opens a print job, which creates a blank page, prints a bitmap on the page, then sends the current page to the printer or spooler and closes the job:

```
ulong Job

// Begin a new job and a new page.
Job = PrintOpen( )

// Print the bitmap in its original size.
PrintBitmap(Job, d:\PB\BITMAP1, 5,10, 0,0)

// Send the page to the printer and close Job.
PrintClose(Job)
```

See also PrintCancel
PrintOpen

PrintDataWindow

Description

Prints the contents of a DataWindow control or DataStore as a print job.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✓

Syntax

PrintDataWindow (*printjobnumber*, *dwcontrol*)

Argument	Description
printjobnumber	The number the PrintOpen function assigned to the print job
dwcontrol	The name of the DataWindow control, child DataWindow, or DataStore containing the DataWindow object you want to print

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is null, PrintDataWindow returns null.

Usage

Do not use PrintDataWindow with any Print functions except PrintOpen and PrintClose.

When you use PrintDataWindow with PrintOpen and PrintClose, you can print several DataWindows in one print job. The information in each DataWindow control starts printing on a new page.

Required third-party software
You must install the FieldSoftware PrinterCE SDK before you can use print methods in PocketBuilder applications deployed to a Pocket PC device or emulator. An evaluation version of this software is available from the FieldSoftware Web site at http://www.fieldsoftware.com.

When you print a DataWindow using PrintDataWindow, PocketBuilder uses the fonts and layout specified in the computer's printer setup, not the fonts and layout specified in the DataWindow. The PrintDefineFont and PrintSetFont methods also have no effect.

When the DataWindow's presentation style is RichTextEdit, each row begins a new page in the printed output.

For information on skipping individual pages with return codes in the PrintPage event, see the Print function.

Examples

These statements send the contents of three DataWindow controls to the current printer in a single print job:

```
long job
job = PrintOpen( )
// Each DataWindow starts printing on a new page.
PrintDataWindow(job, dw_EmpHeader)
PrintDataWindow(job, dw_EmpDetail)
PrintDataWindow(job, dw_EmpDptSum)
PrintClose(job)
```

See also

Print
PrintClose
PrintOpen

PrintDefineFont

Description

Creates a numbered font definition that consists of a font supported by your printer and a set of font properties. You can use the font number in the PrintSetFont or PrintText functions. You can define up to eight fonts at a time.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✓

Syntax

PrintDefineFont (*printjobnumber*, *fontnumber*, *facename*, *height*, *weight*, *fontpitch*, *fontfamily*, *italic*, *underline*)

Argument	Description
printjobnumber	The number the PrintOpen function assigned to the print job.
fontnumber	The number (1 to 8) you want to assign to the font.
facename	A string whose value is the name of a typeface supported by your printer (for example, Courier 10Cpi).
height	An integer whose value is the height of the type in thousandths of an inch (for example, 250 for 18-point 10Cpi) or a negative number representing the point size (for example, -18 for 18-point). Specifying the point size is more exact; the height in thousandths of an inch only approximates the point size.
weight	The stroke weight of the type. Normal weight is 400 and bold is 700.

Argument	Description
fontpitch	A value of the FontPitch enumerated datatype indicating the pitch of the font: Default! Fixed! Variable!
fontfamily	A value of the FontFamily enumerated datatype indicating the family of the font: AnyFont! Decorative! Modern! Roman! Script! Swiss!
italic	A boolean value indicating whether the font is italic. The default is false (not italic).
underline	A boolean value indicating whether the font is underlined. The default is false (not underlined).

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is null, PrintDefineFont returns null.

Usage

You can use as many as eight fonts in one print job. If you require more than eight fonts in one job, you can call PrintDefineFont again to change the settings for a font number.

Use PrintSetFont to make a font number the current font for the open print job.

Fonts in Microsoft Windows
Although the *fontfamily* argument seems to duplicate information in the font name, Windows uses it along with the font name to identify the correct font or substitute a similar font if the named font is unavailable.

Font names and sizes
Some font names include a size, especially monospaced fonts which include characters per inch. This is the recommended size for the font and does not affect the printed size, which you specify with the *height* argument.

Examples

These statements define a new blank page, and then define print font 1 for *Job* as Courier 10Cpi, 18 point, normal weight, default pitch, Decorative font, with no italic or underline:

```
long Job
Job = PrintOpen()
PrintDefineFont(Job, 1, "Courier 10Cpi", &
   -18, 400, Default!, Decorative!, FALSE, FALSE)
```

See also

PrintClose
PrintOpen
PrintSetFont

PrintGetPrinter

Description

Gets the current printer name.

PocketBuilder	✗
PowerBuilder	✓

Syntax

PrintGetPrinter ( )

Return value

String. Returns current printer information in a tab-delimited format: *printername ~t drivername ~t port*.

PrintGetPrinters

Description

Gets the list of available printers.

PocketBuilder	✗
PowerBuilder	✓

Syntax

PrintGetPrinters ( )

Return value

String. Each printer is listed in the string in the format *printername ~t drivername ~t port ~n*.

PrintLine

Description

Draws a line of a specified thickness between the specified endpoints on the current print page.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✓

Syntax

PrintLine (*printjobnumber*, *x1*, *y1*, *x2*, *y2*, *thickness*)

Argument	Description
printjobnumber	The number the PrintOpen function assigned to the print job
x1	An integer specifying the x coordinate in thousandths of an inch of the start of the line
y1	An integer specifying the y coordinate in thousandths of an inch of the start of the line
x2	An integer specifying the x coordinate in thousandths of an inch of the end of the line
y2	An integer specifying the y coordinate in thousandths of an inch of the end of the line
thickness	An integer specifying the thickness of the line in thousandths of an inch

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is null, PrintLine returns null.

Usage

PrintLine does not change the position of the print cursor, which remains where it was before the function was called.

Required third-party software
You must install the FieldSoftware PrinterCE SDK before you can use print methods in PocketBuilder applications deployed to a Pocket PC device or emulator. An evaluation version of this software is available from the FieldSoftware Web site at http://www.fieldsoftware.com.

Examples

These statements start a new page in a print job and then print a line starting at 0,5 and ending at 7500,5 with a thickness of 10/1000 of an inch:

```
long Job
Job = PrintOpen( )
... // various print commands

// Start a new page.
```

```
PrintPage(Job)
// Print a line at the top of the page
PrintLine(Job,0,5,7500,5,10)
... // Other printing
PrintClose(Job)
```

See also

PrintBitmap
PrintClose
PrintOpen
PrintOval
PrintRect
PrintRoundRect

PrintOpen

Description

Opens a print job and assigns it a number, which you use in other printing statements.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✓

Syntax

PrintOpen ({ *jobname* })

Argument	Description
jobname (optional)	A string specifying a name for the print job. The name is displayed in the Windows Print Manager dialog box and in the Spooler dialog box.

Return value

Long. Returns the job number if it succeeds and -1 if an error occurs. If any argument's value is null, PrintOpen returns null.

Usage

A new print job begins on a new page and the font is set to the default font for the printer. The print cursor is at the upper left corner of the print area.

Use the job number that PrintOpen returns to identify this print job in all subsequent print functions.

Required third-party software
You must install the FieldSoftware PrinterCE SDK before you can use print methods in PocketBuilder applications deployed to a device or emulator. An evaluation version of this software is available from the FieldSoftware Web site at http://www.fieldsoftware.com.

Calling MessageBox after PrintOpen can cause undesirable behavior that is confusing to a user. Calling PrintOpen causes the currently active window in PocketBuilder to be disabled to allow Windows to handle printing. If you display a MessageBox after calling PrintOpen, Windows assigns the active window to be its parent, which is often another application, causing that application to become active.

Balancing *PrintOpen* and *PrintClose*
When you open a print job, you must close (or cancel) it. To avoid hung print jobs, process and close a print job in the same event in which you open it.

Examples

This example opens a job but does not give it a name:

```
ulong li_job
li_job = PrintOpen()
```

This example opens a job and gives it a name:

```
ulong li_job
li_job = PrintOpen("Phone List")
```

See also

Print
PrintBitmap
PrintCancel
PrintClose
PrintDataWindow
PrintDefineFont
PrintLine
PrintOval
PrintPage
PrintRect
PrintRoundRect
PrintSend
PrintSetFont
PrintSetup

PrintText
PrintWidth
PrintX
PrintY

PrintOval

Description

Draws a white oval outlined in a line of the specified thickness on the print page.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✓

Syntax

PrintOval (*printjobnumber*, *x*, *y*, *width*, *height*, *thickness*)

Argument	Description
printjobnumber	The number the PrintOpen function assigned to the print job
x	An integer specifying the x coordinate in thousandths of an inch of the upper-left corner of the oval's bounding box
y	An integer specifying the y coordinate in thousandths of an inch of the upper-left corner of the oval's bounding box
width	An integer specifying the width in thousandths of an inch of the oval's bounding box
height	An integer specifying the height in thousandths of an inch of the oval's bounding box
thickness	An integer specifying the thickness of the line that outlines the oval in thousandths of an inch

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is null, PrintOval returns null.

Usage

The PrintOval, PrintRect, and PrintRoundRect functions draw filled shapes. To print other shapes or text inside the shapes, draw the filled shape first and then add text and other shapes or lines inside it. If you draw the filled shape after other printing functions, it will cover anything inside it. For example, to draw a border around text and lines, draw the oval or rectangular border first and then use PrintLine and PrintText to position the lines and text inside.

PrintOval does not change the position of the print cursor, which remains where it was before the function was called. In general, print functions in which you specify coordinates do not affect the print cursor.

Required third-party software
You must install the FieldSoftware PrinterCE SDK before you can use print methods in PocketBuilder applications deployed to a device or emulator. An evaluation version of this software is available from the FieldSoftware Web site at http://www.fieldsoftware.com.

Examples

This example starts a print job with a new blank page, and then prints an oval that fits in a 1-inch square. The upper-left corner of the oval's bounding box is four inches from the top and three inches from the left edge of the print area. Because its height and width are equal, the oval is actually a circle:

```
long Job



// Define a new blank page.
Job = PrintOpen()


// Print an oval.
PrintOval(Job, 4000, 3000, 1000, 1000, 10)


... // Other printing
PrintClose(Job)
```

See also

PrintBitmap
PrintClose
PrintLine
PrintOpen
PrintRect
PrintRoundRect

PrintPage

Description

Sends the current page to the printer or spooler and begins a new blank page in the current print job.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✓

Syntax

PrintPage (*printjobnumber*)

Argument	Description
printjobnumber	The number the PrintOpen function assigned to the print job

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is null, PrintPage returns null.

Usage

You must install the FieldSoftware PrinterCE SDK before you can use print methods in PocketBuilder applications deployed to a Pocket PC device or emulator. An evaluation version of this software is available from the FieldSoftware Web site at http://www.fieldsoftware.com.

Examples

This example opens a print job with a new blank page, prints a bitmap on the page, and then sends the page to the printer and sets up a new blank page. Finally, the last Print statement prints the company name on the new page:

```
long Job

// Open a job with new blank page.
Job = PrintOpen()

// Print a bitmap on the page.
PrintBitmap(Job, "d:\PB\BITMAP1.BMP", 100,250, 0,0)

// Begin a new page.
PrintPage(Job)

// Print the company name on the new page.
Print(Job, "Sybase Corporation")
```

See also

PrintClose
PrintOpen

PrintRect

Description

Draws a white rectangle with a border of the specified thickness on the print page.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✓

Syntax

PrintRect (*printjobnumber*, *x*, *y*, *width*, *height*, *thickness*)

Argument	Description
printjobnumber	The number the PrintOpen function assigned to the print job
x	An integer specifying the x coordinate in thousandths of an inch of the upper-left corner of the rectangle
y	An integer specifying the y coordinate in thousandths of an inch of the upper-left corner of the rectangle
width	An integer specifying the rectangle's width in thousandths of an inch
height	An integer specifying the rectangle's height in thousandths of an inch
thickness	An integer specifying the thickness of the rectangle's border line in thousandths of an inch

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is null, PrintRect returns null.

Usage

The PrintOval, PrintRect, and PrintRoundRect functions draw filled shapes. To print other shapes or text inside the shapes, draw the filled shape first and then add text and other shapes or lines inside it. If you draw the filled shape after other printing functions, it will cover anything inside it. For example, to draw a border around text and lines, draw the oval or rectangular border first and then use PrintLine and PrintText to position the lines and text inside.

PrintRect does not change the position of the print cursor, which remains where it was before the function was called. In general, print functions in which you specify coordinates do not affect the print cursor.

Required third-party software
You must install the FieldSoftware PrinterCE SDK before you can use print methods in PocketBuilder applications deployed to a device or emulator. An evaluation version of this software is available from the FieldSoftware Web site at http://www.fieldsoftware.com.

Examples

These statements open a print job with a new page and draw a 1-inch square with a line thickness of 1/8 of an inch. The square's upper left corner is four inches from the left and three inches from the top of the print area:

```
long Job
// Define a new blank page.
Job = PrintOpen()
// Print the rectangle on the page.
PrintRect(Job, 4000,3000, 1000,1000, 125)
... // Other printing
PrintClose(Job)
```

See also

PrintBitmap
PrintClose
PrintLine
PrintOpen
PrintOval
PrintRoundRect

PrintRoundRect

Description

Draws a white rectangle with rounded corners and a border of the specified thickness on the print page.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✓

Syntax

PrintRoundRect (*printjobnumber*, *x*, *y*, *width*, *height*, *xradius*, *yradius*, *thickness*)

Argument	Description
printjobnumber	The number the PrintOpen function assigned to the print job
x	An integer specifying the x coordinate in thousandths of an inch of the upper-left corner of the rectangle
y	An integer specifying the y coordinate in thousandths of an inch of the upper-left corner of the rectangle
width	An integer specifying the rectangle's width in thousandths of an inch
height	An integer specifying the rectangle's height in thousandths of an inch

Argument	Description
xradius	An integer specifying the x radius of the corner rounding
yradius	An integer specifying the y radius of the corner rounding
thickness	An integer specifying the thickness of the rectangle's border line in thousandths of an inch

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is null, PrintRoundRect returns null.

Usage

The PrintOval, PrintRect, and PrintRoundRect functions draw filled shapes. To print other shapes or text inside the shapes, draw the filled shape first and then add text and other shapes or lines inside it. If you draw the filled shape after other printing functions, it will cover anything inside it. For example, to draw a border around text and lines, draw the oval or rectangular border first and then use PrintLine and PrintText to position the lines and text inside.

PrintRoundRect does not change the position of the print cursor, which remains where it was before the function was called. In general, print functions in which you specify coordinates do not affect the print cursor.

Required third-party software
You must install the FieldSoftware PrinterCE SDK before you can use print methods in PocketBuilder applications deployed to a device or emulator. An evaluation version of this software is available from the FieldSoftware Web site at http://www.fieldsoftware.com.

Examples

This example starts a new print job, which begins a new page, and prints a rectangle with rounded corners as a page border. Then it closes the print job, which sends the page to the printer.

The rectangle is 6 1/4 inches wide by 9 inches high and its upper corner is one inch from the top and one inch from the left edge of the print area. The border has a line thickness of 1/8 of an inch and the corner radius is 300:

```
long Job

// Define a new blank page.
Job = PrintOpen()

// Print a RoundRectangle on the page.
PrintRoundRect(Job, 1000,1000, 6250,9000, &
   300,300, 125)

// Send the page to the printer.
PrintClose(Job)
```

See also

PrintBitmap
PrintClose
PrintLine
PrintOpen
PrintOval
PrintRect

PrintScreen

Description

Prints the screen image as part of a print job.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✓

Syntax

PrintScreen (*printjobnumber*, *x*, *y* {, *width*, *height* })

Argument	Description
printjobnumber	The number the PrintOpen function assigns to the print job.
x	An integer whose value is the x coordinate on the page, in thousandths of an inch, of the upper-left corner of the screen image.
y	An integer whose value is the y coordinate on the page, in thousandths of an inch, of the upper-left corner of the screen image.
width (optional)	The integer width of the printed screen in thousandths of an inch. If you omit *width*, PocketBuilder prints the screen at its original width. If you specify *width*, you must also specify *height*.
height (optional)	The integer height of the printed screen in thousandths of an inch. If you omit *height*, PocketBuilder prints the screen at its original height.

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is null, PrintScreen returns null.

Usage

You must install the FieldSoftware PrinterCE SDK before you can use print methods in PocketBuilder applications deployed to a device or emulator. An evaluation version of this software is available from the FieldSoftware Web site at http://www.fieldsoftware.com.

Examples

This statement prints the current screen image in its original size at location 500, 1000:

```
long Job
Job = PrintOpen()
PrintScreen(Job, 500,1000)
PrintClose(Job)
```

See also

Print
PrintClose
PrintOpen

PrintSend

Description

Sends an arbitrary string of characters to the printer. PrintSend is usually used for sending escape sequences that change the printer's setup.

PocketBuilder	✗
PowerBuilder	✓

Obsolete function
PrintSend is an obsolete function. The ability to use this function is dependent upon the printer driver.

Syntax

PrintSend (*printjobnumber*, *string* {, *zerochar* })

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is null, PrintSend returns null.

PrintSetFont

Description

Designates a font to be used for text printed with the Print function. You specify the font by number. Use PrintDefineFont to associate a font number with the desired font, a size, and a set of properties.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✓

Syntax

PrintSetFont (*printjobnumber*, *fontnumber*)

Argument	Description
printjobnumber	The number the PrintOpen function assigned to the print job
fontnumber	The number (1 to 8) of a font defined for the job in PrintDefineFont or 0 (the default font for the printer)

Return value

Integer. Returns the character height of the current font if it succeeds and -1 if an error occurs. If any argument's value is null, PrintSetFont returns null.

Examples

This example starts a new print job and specifies that font number 2 is Courier, 18 point, bold, default pitch, in modern font, with no italic or underline. The PrintSetFont statement sets the current font to font 2. Then the Print statement prints the company name:

```
long Job

// Start a new print job and a new page.
Job = PrintOpen()

// Define the font for Job.
PrintDefineFont(Job, 2, "Courier 10Cps", &
   250, 700, Default!, Modern!, FALSE, FALSE)

// Set the font for Job.
PrintSetFont(Job, 2)

// Print the company name in the specified font.
Print(Job,"Sybase Corporation")
```

See also

PrintDefineFont
PrintOpen

PrintSetPrinter

Description

Sets the printer to use for the next print function call. This function does not affect open jobs.

PocketBuilder	✗
PowerBuilder	✓

Syntax

PrintSetPrinter (*printername*)

Return value

Integer. Returns 1 if the function succeeds and -1 if an error occurs.

PrintSetSpacing

Description

Sets the factor that PocketBuilder uses to calculate line spacing.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✓

Syntax

PrintSetSpacing (*printjobnumber*, *spacingfactor*)

Argument	Description
printjobnumber	The number the PrintOpen function assigned to the print job.
spacingfactor	The number by which you want to multiply the character height to determine the vertical line-to-line spacing. The default is 1.2.

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is null, PrintSetSpacing returns null.

Usage

Line spacing in PocketBuilder is proportional to character height. The default line spacing is 1.2 times the character height. When Print starts a new line, it sets the x coordinate of the cursor to 0 and increases the y coordinate by the current line spacing. The PrintSetSpacing function lets you specify a new factor to be multiplied by the character height for an open print job.

Examples

These statements start a new print job and set the vertical spacing factor to 1.5 (one and a half spacing):

```
long Job
```

```
// Define a new blank page.
Job = PrintOpen()

// Set the spacing factor.
PrintSetSpacing(Job, 1.5)
```

See also PrintOpen

PrintSetup

Description Calls the Printer Setup dialog box provided by the system printer driver and lets the user specify settings for the printer.

PocketBuilder	✗
PowerBuilder	✓

Syntax **PrintSetup** ( )

Return value Integer. Returns 1 if it succeeds and -1 if an error occurs.

PrintSetupPrinter

Displays the printer setup dialog box for PowerBuilder applications. Bypasses the printer setup dialog box for PocketBuilder applications.

To	Use
Display the printer setup dialog box	Syntax 1
Bypass the printer setup dialog box	Syntax 2

Syntax 1 For displaying the printer setup dialog box

Description Displays the printer setup dialog box.

PocketBuilder	✗
PowerBuilder	✓

Syntax **PrintSetupPrinter** ( )

Return value

Integer. Returns 1 if the function succeeds, 0 for cancel, -1 if an error occurs.

Usage

Although you can call PrintSetupPrinter without any arguments in PocketBuilder, it does not change the current printer setup or display the printer setup dialog box.

Syntax 2 For bypassing the printer setup dialog box

Description

Bypasses the printer setup dialog box.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✗

Syntax

PrintSetupPrinter (Ipaddr, path, type, port, rate)

Argument	Description
Ipaddr	A string setting the IP address of a networked printer. If you use the path name instead of an IP address, you should enter an empty string for this argument.
path	A string for the network path of a host PC with a shared printer. If you use the IP address instead of a network path, you should enter an empty string for this argument.
type	A long value specifying the printer type. For example, you specify a a Hewlett-Packard PCL compatible laser printer with a value of 5. Printer types and their corresponding values are specified in the FieldSoftware *PrinterCE Developer's Guide*.
port	A long value specifying the printer port. Example values for different ports are: 0 for COM1, 9 for a shared printer on a host PC, 10 for a printer with its own IP address. Available ports and their corresponding values are specified in the FieldSoftware *PrinterCE Developer's Guide*.
rate	A long value specifying the baud rate. You can select the following values for available baud rates: • 0 for 4800 baud or for networked printers • 1 for 9600 baud printers • 2 for 19200 baud printers • 3 for 34800 baud printers • 4 for 57600 baud printers • 5 for 115200 baud printers

Return value

Integer. Returns 1 if the function succeeds, -1 if an error occurs.

Usage

You can enter either an IP address for the printer you want to set up or a complete network path to the printer. Although you must enter string values for the first two arguments of this function, one of these arguments should take an empty string (“ ”) for a value.

If you use the *Ipaddr* argument to specify the printer, you would typically use a value of 10 for the *port* argument. If you use the *path* argument to specify the printer, you would typically enter a value of 9 for the *port* argument.

Required third-party software
You must install the FieldSoftware PrinterCE SDK before you can use print methods in PocketBuilder applications deployed to a device or emulator. An evaluation version of this software is available from the FieldSoftware Web site at http://www.fieldsoftware.com.

Examples

The following example prints to an Epson Stylus compatible network printer:

```
li_rtn = PrintSetupPrinter ("10.18.61.20", "", 4, 10, 0)
```

This example prints to an HP Laser Jet printer with a defined path at a 9600 baud rate:

```
PrintSetupPrinter ("", "\\MyPC\HPLaserJ", 5, 9, 1)
```

PrintText

Description

Prints a single line of text starting at the specified coordinates.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✓

Syntax

PrintText (*printjobnumber*, *string*, *x*, *y* {, *fontnumber* })

Argument	Description
printjobnumber	The number the PrintOpen function assigned to the print job.
string	A string whose value is the text you want to print.
x	An integer specifying the x coordinate in thousandths of an inch of the beginning of the text.

Argument	Description
y	An integer specifying the y coordinate in thousandths of an inch of the beginning of the text.
fontnumber (optional)	The number (1 to 8) of a font defined for the job by using the PrintDefineFont function or 0 (the default font for the printer). If you omit *fontnumber*, the text prints in the current font for the print job.

Return value

Integer. Returns the x coordinate of the new cursor location (that is, the value of the parameter *x* plus the width of the text) if it succeeds. PrintText returns -1 if an error occurs. If any argument's value is null, PrintText returns null.

Usage

PrintText does change the position of the print cursor, unlike the other print functions for which you specify coordinates. The print cursor moves to the end of the printed text. PrintText also returns the x coordinate of the print cursor. You can use the return value to determine where to begin printing additional text.

PrintText does not change the print cursor's y coordinate, which is its vertical position on the page.

Required third-party software
You must install the FieldSoftware PrinterCE SDK before you can use print methods in PocketBuilder applications deployed to a device or emulator. An evaluation version of this software is available from the FieldSoftware Web site at http://www.fieldsoftware.com.

Examples

These statements start a new print job and then print `PocketBuilder` in the current font 3.7 inches from the left edge at the top of the page (location 3700,10):

```
long Job

// Define a new blank page.
Job = PrintOpen()

// Print the text.
PrintText(Job,"PocketBuilder", 3700, 10)
... // Other printing
PrintClose(Job)
```

The following statements define a new blank page and then print `Confidential` in bold (as defined for font number 3), centered at the top of the page:

```
long Job

// Start a new job and a new page.
Job = PrintOpen()

// Define the font.
PrintDefineFont(Job, 3, &
   "Courier 10Cps", 250,700, &
      Default!, AnyFont!, FALSE, FALSE)

// Print the text.
PrintText(Job, "Confidential", 3700, 10, 3)
... // Other printing
PrintClose(Job)
```

This example prints four lines of text in the middle of the page. The coordinates for PrintText establish a new vertical position for the print cursor, which the subsequent Print functions use and increment. The first Print function uses the x coordinate returned by PrintText to continue the first line. The rest of the Print functions print additional lines of text, after tabbing to the x coordinate used initially by PrintText. In this example, each Print function increments the y coordinate so that the following Print function starts a new line:

```
long Job

// Start a new job and a new page.
Job = PrintOpen()

// Print the text.
x =  PrintText(Job,"The material ", 2000, 4000)
Print(Job, x, " in this report")
Print(Job, 2000, "is confidential and should not")
Print(Job, 2000, "be disclosed to anyone who")
Print(Job, 2000, "is not at this meeting.")
... // Other printing
PrintClose(Job)
```

See also

Print
PrintClose
PrintOpen

PrintWidth

Description

Determines the width of a string using the current font of the specified print job.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✓

Syntax

PrintWidth (*printjobnumber*, *string*)

Argument	Description
printjobnumber	The number the PrintOpen function assigned to the print job
string	A string whose value is the text for which you want to determine the width

Return value

Integer. Returns the width of *string* in thousandths of an inch using the current font of *printjobnumber* if it succeeds and -1 if an error occurs. If any argument's value is null, PrintWidth returns null. If the returned width exceeds the maximum integer limit (+32767), PrintWidth returns -1.

Examples

These statements define a new blank page and then set *W* to the length of the string `PowerBuilder` in the current font and then use the length to position the next text line:

```
long Job
int W

// Start a new print job.
Job = PrintOpen()

// Determine the width of the text.
W = PrintWidth(Job,"PowerBuilder")

// Use the width to get the next print position.
Print(Job, W - 500, "Features List")
```

See also

PrintClose
PrintOpen

PrintX

Description

Reports the x coordinate of the print cursor.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✓

Syntax

PrintX (*printjobnumber*)

Argument	Description
printjobnumber	The number the PrintOpen function assigned to the print job

Return value

Integer. Returns the x coordinate of the print cursor if it succeeds and -1 if an error occurs. If any argument's value is null, PrintX returns null.

Examples

These statements set *LocX* to the x coordinate of the cursor and print `End of Report` an inch beyond that location:

```
integer LocX
long Job

Job = PrintOpen()
... //Print statements
LocX = PrintX(Job)
Print(LocX+1000, "End of Report")
```

See also

PrintY

PrintY

Description

Reports the y coordinate of the print cursor.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✓

Syntax

PrintY (*printjobnumber*)

Argument	Description
printjobnumber	The number the PrintOpen function assigned to the print job

Return value

Integer. Returns the y coordinate of the cursor if it succeeds and -1 if an error occurs. If any argument's value is null, PrintY returns null.

Examples

These statements print a bitmap one inch below the location of the print cursor:

```
integer LocX, LocY
long Job


Job = PrintOpen()
... //Print statements
LocX = PrintX(Job)
LocY = PrintY(Job) + 1000
PrintBitmap(Job, "CORP.BMP", LocX, LocY, 1000,1000)
```

See also

PrintX

ProfileInt

Description

Obtains the integer value of a setting in the profile file for your application.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

File format
ProfileInt can read either ANSI or Unicode files.

Syntax

ProfileInt (*filename*, *section*, *key*, *default*)

Argument	Description
filename	A string whose value is the name of the profile file. If you do not specify a full path, ProfileInt uses the operating system's standard file search order to find the file.
section	A string whose value is the name of a group of related values in the profile file. In the file, section names are in square brackets. Do not include the brackets in *section*. *Section* is not case sensitive.
key	A string specifying the setting name in *section* whose value you want. The setting name is followed by an equal sign in the file. Do not include the equal sign in *key*. *Key* is not case sensitive.
default	An integer value that ProfileInt will return if *filename* is not found, if *section* or *key* does not exist in *filename,* or if the value of *key* cannot be converted to an integer.

Return value

Integer. Returns *default* if *filename* is not found, *section* is not found in *filename*, or *key* is not found in *section*, or the value of *key* is not an integer. Returns -1 if an error occurs. If any argument's value is null, ProfileInt returns null.

Usage

Use ProfileInt or ProfileString to get configuration settings from a profile file that you have designed for your application.

You can use SetProfileString to change values in the profile file to customize your application's configuration during execution. Before you make changes, you can use ProfileInt and ProfileString to obtain the original settings so you can restore them when the user exits the application.

Windows registry
ProfileInt can also be used to obtain configuration settings from the Windows system registry. For information on how to use the system registry, see the discussion of initialization files and the Windows registry in the *Resource Guide*.

Examples

These examples use a file called *PROFILE.INI*, which contains the following:

```
[Pb]
Maximized=1
[security]
Class=7
```

This statement returns the integer value for the keyword Maximized in section PB of file *PROFILE.INI*. If there were no PB section or no Maximized keyword in the PB section, it would return 3:

```
ProfileInt("C:\PROFILE.INI", "PB", "maximized", 3)
```

The following statements display a MessageBox if the integer value for the Class setting in section Security of file *C:\PROFILE.INI* is less than 10. The default security setting is 6 if the profile file is not found or does not contain a Class setting:

```
IF ProfileInt("C:\PROFILE.INI", "Security", &
   "Class", 6) < 10 THEN
   // Class is < 10
   MessageBox("Warning", "Access Denied")
ELSE
 ... // Some processing
END IF
```

See also

ProfileString
SetProfileString
ProfileInt method for DataWindows in the *DataWindow Reference*

ProfileString

Description

Obtains the string value of a setting in the profile file for your application.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

File format
ProfileString can read either ANSI or Unicode files.

Syntax

ProfileString (*filename*, *section*, *key*, *default*)

Argument	Description
filename	A string whose value is the name of the profile file. If you do not specify a full path, ProfileString uses the operating system's standard file search order to find the file.
section	A string whose value is the name of a group of related values in the profile file. In the file, section names are in square brackets. Do not include the brackets in *section*. *Section* is not case sensitive.
key	A string specifying the setting name in *section* whose value you want. The setting name is followed by an equal sign in the file. Do not include the equal sign in *key*. *Key* is not case sensitive.
default	A string value that ProfileString will return if *filename* is not found, or if *section* or *key* does not exist in *filename*.

Return value

String, with a maximum length of 4096 characters. Returns the string from *key* within *section* within *filename*. If *filename* is not found, *section* is not found in *filename*, or *key* is not found in *section*, ProfileString returns *default*. If *key* does not have a value, ProfileString returns the empty string ("").

Usage

Use ProfileInt or ProfileString to get configuration settings from a profile file that you have designed for your application.

You can use SetProfileString to change values in the profile file to customize your application's configuration during execution. Before you make changes, you can use ProfileInt and ProfileString to obtain the original settings so you can restore them when the user exits the application.

Windows registry
ProfileString can also be used to obtain configuration settings from the Windows system registry. For information on how to use the system registry, see the discussion of initialization files and the Windows registry in the *Resource Guide*.

Examples

These examples use a file called *PROFILE.INI*, which contains the following lines. Quotes around string values in the INI file are optional:

```
[Employee]
Name=Smith

[Dept]
Name=Marketing
```

This statement returns the string contained in keyword Name in section Employee in file *C:\PROFILE.INI* and returns None if there is an error. In the example, the return value is Smith:

```
ProfileString("C:\PROFILE.INI", "Employee", &
   "Name", "None")
```

The following statements open w_marketing if the string in the keyword Name in section Department of file *C:\PROFILE.INI* is Marketing:

```
IF ProfileString("C:\PROFILE.INI", "Department", &
   "Name", "None") = "Marketing" THEN
   Open(w_marketing)
END IF
```

See also

ProfileInt
SetProfileString
ProfileString method for DataWindows in the *DataWindow Reference*

Rand

Description

Obtains a random whole number between 1 and a specified upper limit.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

Rand (*n*)

Argument	Description
n	The upper limit of the range of random numbers you want returned. The lower limit is always 1. The upper limit is 32,767.

Return value

A numeric datatype, the datatype of *n*. Returns a random whole number between 1 and *n* inclusive. If *n* is null, Rand returns null.

Usage

The sequence of numbers generated by repeated calls to the Rand function is a pseudorandom sequence. You can control whether the sequence is different each time your application runs by calling the Randomize function to initialize the random number generator.

Examples

This statement returns a random whole number between 1 and 10:

```
Rand(10)
```

See also

Randomize

Randomize

Description

Initializes the random number generator so that the Rand function begins a new series of pseudorandom numbers.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

Randomize (*n*)

Argument	Description
n	The starting value (seed) for the random number generator. When *n* is 0, PocketBuilder takes the seed from the system clock and begins a nonrepeatable sequence. A nonzero number generates a different but repeatable sequence for each seed value. *n* cannot exceed 32,767.

Return value

Integer. If *n* is null, Randomize returns null. The return value is never used.

Usage

The sequence of numbers generated by repeated calls to the Rand function is a computer-generated pseudorandom sequence. You can use the Randomize function to initialize the random number generator with a value from the system clock, or some other changing value, so that the sequence is always different. For testing purposes, you can select a specific seed value, which you can reuse to make the pseudorandom sequence repeatable each time you run the application.

Include Randomize in the script for the Open event in the application.

Examples

This statement sets the seed for the random number generator to 0 so that calls to Rand generate a new sequence each time the script is run:

```
Randomize(0)
```

This statement sets the seed for the random number generator to 4 so that calls to Rand repeat a specific sequence each time the random number generator is initialized:

```
Randomize(4)
```

See also

Rand

Read

For OLE stream objects Reads data from an opened OLE stream object.

To	Use
Read data into a string	Syntax 1
Read data into a character array or blob	Syntax 2

For FileDirect objects Reads a file that you open using the FileDirect object.

To	Use
Read data into an array	Syntax 3
Read data into a blob	Syntax 4

Syntax 1 For reading into a string

Description

Reads data from an OLE stream object into a string.

PocketBuilder	✗
PowerBuilder	✓

Applies to

OLEStream objects

Syntax

olestream.**Read** (*variable* {, *stopforline* })

Return value

Integer. Returns the number of characters or bytes read. If an end-of-file mark (EOF) is encountered before any characters are read, Read returns -100. Read returns a negative integer if an error occurs.

Syntax 2 For character arrays or blobs

Description

Reads data from an OLE stream object into a character array or blob.

PocketBuilder	✗
PowerBuilder	✓

Applies to

OLEStream objects

Syntax

olestream.**Read** (*variable* {, *maximumread* })

Return value

Integer. Returns 0 if it succeeds and a negative integer if an error occurs.

Syntax 3 For reading data into an array

Description

Reads data from an open file into an array.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✗

Applies to FileDirect objects

Syntax Integer *instancename*.Read (*data[ ]*, *bytesrequested*, *bytesread*)

Argument	Description
instancename	Name of the instance of the FileDirect object
data[ ]	An array of unsignedlong datatypes to contain the data that you read from a file
bytesrequested	Integer for the number of bytes that you want to read in the open file
bytesread	Integer for storing the number of bytes read in the file

Return value Integer. Returns 1 for success and a negative number for any error.

Usage Use this function to read a file that you open with the FileDirect object in read mode. The FileDirect object supports only the synchronous style of file output; further file-related commands cannot be called until after the Read function is fully processed or an error in reading the file is caught.

Examples The following example calls the FileDirect user object nvo_fileDirect to open a file, read some data, store the data in an array, and close the file:

```
Integer li_ret, li_AmountRead
Unsignedlong li_data [ ]
li_ret = nvo_fileDirect.Open ("COM8:", stgReadWrite!)
li_ret = nvo_fileDirect.Read (li_data[], 100,
    li_amountRead)
li_ret = nvo_fileDirect.Close ( )
```

See also Write

Syntax 4 For reading data into a blob

Description Reads data from an open file into a blob.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✗

Applies to FileDirect objects

Syntax Integer *instancename*.Read (*bdata*, *bytesrequested*, *bytesread*)

Argument	Description
instancename	Name of the instance of the FileDirect object

Argument	Description
bdata	A blob variable to hold the data that you read from a file
bytesrequested	Integer for the number of bytes that you want to read in the open file
bytesread	Integer for for storing the number of bytes read in the file

Return value

Integer. Returns 1 for success and a negative number for any error.

Usage

Use this function to read a file that you open with the FileDirect object in read mode. The FileDirect object supports only the synchronous style of file output; further file-related commands cannot be called until after the Read function is successfully processed or until an error in reading the file is caught.

Examples

The following example calls the FileDirect user object nvo_fileDirect to open a file, read some data, store the data in a blob variable, and close the file:

```
Integer li_ret, li_AmountRead
Blob lb_data
li_ret = nvo_fileDirect.Open ("MyDoc.txt", stgRead!)
li_ret = nvo_fileDirect.Read (lb_data, 100,
    li_amountRead)
li_ret = nvo_fileDirect.Close ( )
```

See also

Open
Seek
Write

Real

Description

Converts a string value to a real datatype or obtains a real value that is stored in a blob.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

Real (*stringorblob*)

Argument	Description
stringorblob	The string whose value you want returned as a real value or a blob in which the first value is the real value. The rest of the contents of the blob is ignored. *Stringorblob* can also be an Any variable containing a string or blob.

Return value

Real. Returns the value of *stringorblob* as a real. If *stringorblob* is not a valid PowerScript number or is an incompatible datatype, Real returns 0. If *stringorblob* is null, Real returns null.

Examples

This statement returns 24 as a real:

```
Real("24")
```

This statement returns the contents of the SingleLineEdit sle_Temp as a real:

```
Real(sle_Temp.Text)
```

The following example, although of no practical value, illustrates how to assign real values to a blob and how to use Real to extract those values. The two BlobEdit statements store two real values in the blob, one after the other. In the statements that use Real to extract the values, you have to know where the beginning of each real value is. Specifying the correct length in BlobMid is not important because the Real function knows how many bytes to evaluate:

```
blob{20} lb_blob
real r1, r2
integer len1, len2

len1 = BlobEdit(lb_blob, 1, 32750E0)
len2 = BlobEdit(lb_blob, len1, 43750E0)

// Extract the real value at the beginning and
// ignore the rest of the blob
r1 = Real(lb_blob)
// Extract the second real value stored in the blob
r2 = Real(BlobMid(lb_blob, len1, len2 - len1))
```

See also

Double
Integer
Long
Real method for DataWindows in the *DataWindow Reference*

ReceiveFromInfrared

Description

Receives items over an infrared link and distributes them to destination folders.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✗

Applies to

POOM objects

Syntax

Integer *objectname*.ReceiveFromInfrared ( )

Argument	Description
objectname	The name of the POOM object

Return value

Integer. Returns 1 for success and one of the following negative values if an error occurs:

-1	Unspecified error
-2	Cannot connect to the repository or a required internal subobject failed to connect to the repository
-3	Cannot log in to the repository
-4	Incorrect input argument
-5	Action cannot be performed
-6	The object identifier (OID) is not in the repository
-7	Feature is not implemented yet
-8	No matching entries found for the criteria

Usage

A user must be logged in to a POOM object to receive an infrared queue. Calling ReceiveFromInfrared turns on the infrared receivers and places the POOM objects it receives into the correct appointment, task, and contact categories in the POOM repository.

Examples

The following example retrieves items from an infrared queue:

```
li_rtn = po_1.ReceiveFromInfrared()
```

See also

AddToInfraredQueue
SendToInfrared

RegistryDelete

Description

Deletes a key or a value for a key in the Windows system registry.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

RegistryDelete (*key*, *valuename*)

Argument	Description
key	A string whose value is the key in the system registry you want to delete or whose value you want to delete. To uniquely identify a key, specify the list of parent keys above it in the hierarchy, starting with the root key. The keys in the list are separated by backslashes.
valuename	A string containing the name of a value in the registry. If the specified key does not have a subkey, specifying an empty string deletes the key and its named values.

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs.

Usage

For more information about entries in the system registry, see RegistrySet.

Examples

This statement deletes the value name Title and its associated value from the registry. The key is not deleted:

```
RegistryDelete( &
 "HKEY_LOCAL_MACHINE\Software\MyApp.Settings\Fonts", &
 "Title")
```

See also

RegistryGet
RegistryKeys
RegistrySet
RegistryValues

RegistryGet

Description

Gets a value from the Windows system registry.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

RegistryGet (*key*, *valuename*, { *valuetype* }, *valuevariable*)

Argument	Description
key	A string whose value names the key in the system registry whose value you want. To uniquely identify a key, specify the list of parent keys above it in the hierarchy, starting with the root key. The keys in the list are separated by backslashes.
valuename	A string containing the name of a value in the registry. Each key can have one unnamed value and several named values. For the unnamed value, specify an empty string.
valuetype	A value of the RegistryValueType enumerated datatype identifying the datatype of a value in the registry. Values are: • RegString!—A null-terminated string • RegExpandString!—A null-terminated string that contains unexpanded references to environment variables • RegBinary!—Binary data • ReguLong!—A 32-bit number • ReguLongBigEndian!—A 32-bit number • RegLink!—A Unicode symbolic link • RegMultiString!—An unbounded array of strings
valuevariable	A variable corresponding to the datatype of *valuetype* in which you want to store the value obtained from the system registry for the specified key and value name.

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs. An error is returned if the datatype of *valuevariable* does not correspond to the datatype specified in *valuetype*.

Usage

Long string values (more than 2048 bytes) should be stored as files and the file name stored in the registry.

For more information about keys and value names in the system registry, see RegistrySet.

Examples

This statement obtains the value for the name Title and stores it in the string *ls_titlefont*:

```
string ls_titlefont
RegistryGet( &
 "HKEY_LOCAL_MACHINE\Software\MyApp.Settings\Fonts", &
"Title", RegString!, ls_titlefont)
```

This statement obtains the value for the name NameOfEntryNum and stores it in the long *ul_num*:

```
ulong ul_num
RegistryGet( &
 "HKEY_USERS\MyApp.Settings\Fonts", &
 "NameOfEntryNum", RegULong!, ul_num)
```

See also

RegistryDelete
RegistryKeys
RegistrySet
RegistryValues

RegistryKeys

Description

Obtains a list of the keys that are child items (subkeys) one level below a key in the Windows system registry.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

RegistryKeys (*key*, *subkeys*)

Argument	Description
key	A string whose value names the key in the system registry whose subkeys you want. To uniquely identify a key, specify the list of parent keys above it in the hierarchy, starting with the root key. The keys in the list are separated by backslashes.
subkeys	An array variable of strings in which you want to store the subkeys. If the array is variable size, its upper bound will reflect the number of subkeys found. If the array is fixed size, it must be large enough to hold all the subkeys. However, there will be no way to know how many subkeys were actually found.

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs.

Usage

For more information about entries in the system registry, see RegistrySet.

Examples

This example obtains the subkeys associated with the key *HKEY_CLASSES_ROOT\MyApp*. The subkeys are stored in the variable-size array *ls_subkeylist*:

```
string ls_subkeylist[]
integer li_rtn
li_rtn = RegistryKeys("HKEY_CLASSES_ROOT\MyApp", &
    ls_subkeylist)
IF li_rtn = -1 THEN
    ... // Error processing
END IF
```

See also

RegistryDelete
RegistryGet
RegistrySet
RegistryValues

RegistrySet

Description

Sets the value for a key and value name in the system registry. If the key or value name does not exist, RegistrySet creates a new key or name and sets its value.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

RegistrySet (*key*, *valuename*, *valuetype, value*)

Argument	Description
key	A string whose value names the key in the system registry whose value you want to set. To uniquely identify a key, specify the list of parent keys above it in the hierarchy, starting with the root key. The keys in the list are separated by backslashes. If *key* does not exist in the registry, RegistrySet creates a new key. To create a *key* without a named value, specify an empty string for *valuename*.
valuename	A string containing the name of a value in the registry. Each key may have several named values. To specify the unnamed value, specify an empty string. If *valuename* does not exist in the registry, RegistrySet causes a new name to be created for *key*.
valuetype	A value of the RegistryValueType enumerated datatype identifying the datatype of a value in the registry. Values are: • RegString!—A null-terminated string • RegExpandString!—A null-terminated string that contains unexpanded references to environment variables • RegBinary!—Binary data • ReguLong!—A 32-bit number • ReguLongBigEndian!—A 32-bit number • RegLink!—A Unicode symbolic link • RegMultiString!—An unbounded array of strings
value	A variable corresponding to the datatype of *valuetype* containing a value to be set in the registry.

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs. An error is returned if the datatype of *valuevariable* does not correspond to the datatype specified in *valuetype*.

Usage

Long string values (more than 2048 bytes) should be stored as files and the file name stored in the registry.

Item	Description

Key	An element in the registry. A key is part of a tree of keys, descending from one of the predefined root keys. Each key is a subkey or child of the parent key above it in the hierarchy.
	There are four root strings:
	• HKEY_CLASSES_ROOT
	• HKEY_LOCAL_MACHINE
	• HKEY_USERS
	• HKEY_CURRENT_USER
	A key is uniquely identified by the list of parent keys above it. The keys in the list are separated by slashes, as shown in these examples:
	`HKEY_CLASSES_ROOT\Sybase.Application`
	`HKEY_USERS\MyApp\Display\Fonts`

Value name	The name of a value belonging to the key. A key can have one unnamed value and one or more named values.
Value type	A value identifying the datatype of a value in the registry.
Value	A value associated with a value name or an unnamed value. Several string, numeric, and binary datatypes are supported by the registry.

Examples

This example sets a value for the key Fonts and the value name Title:

```
RegistrySet( &
 "HKEY_LOCAL_MACHINE\Software\MyApp\Fonts", &
 "Title", RegString!, sle_font.Text)
```

This statement sets a value for the key Fonts and the value name NameOfEntryNum:

```
ulong ul_num
RegistrySet( &
 "HKEY_USERS\MyApp.Settings\Fonts", &
 "NameOfEntryNum", RegULong!, ul_num)
```

See also

RegistryDelete
RegistryGet
RegistryKeys
RegistryValues

RegistryValues

Description

Obtains the list of named values associated with a key.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

RegistryValues (*key*, *valuename*)

Argument	Description
key	A string whose value is the key in the system registry for which you want the values of its subkeys. To uniquely identify a key, specify the list of parent keys above it in the hierarchy, starting with the root key. The keys in the list are separated by backslashes.
valuename	An array variable of strings in which you want to store the names. If the array is variable size, its upper bound will reflect the number of named values found. If the array is fixed size, it must be large enough to hold all the names. However, there will be no way to know how many names were actually found.

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs.

Usage

For more information about entries in the system registry, see RegistrySet.

Examples

This example gets the value names associated with the key Fonts and stores them in the array *ls_valuearray*:

```
string ls_valuearray[]
RegistryValues( &
  "HKEY_LOCAL_MACHINE\Software\MyApp.Settings\Fonts",&
    ls_valuearray)
```

See also

RegistryDelete
RegistryGet
RegistryKeys
RegistrySet

RelativeDate

Description

Obtains the date that occurs a specified number of days after or before another date.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

RelativeDate (*date*, *n*)

Argument	Description
date	A value of type date
n	An integer indicating a number of days

Return value

Date. Returns the date that occurs *n* days after *date* if *n* is greater than 0. Returns the date that occurs *n* days before *date* if *n* is less than 0. If any argument's value is null, RelativeDate returns null.

Examples

This statement returns 1990-02-10:

```
RelativeDate(1990-01-31, 10)
```

This statement returns 1990-01-21:

```
RelativeDate(1990-01-31,  - 10)
```

See also

DaysAfter
RelativeDate method for DataWindows in the *DataWindow Reference*

RelativeTime

Description

Obtains a time that occurs a specified number of seconds after or before another time within a 24-hour period.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

RelativeTime (*time*, *n*)

Argument	Description
time	A value of type time
n	A long number of seconds

Return value

Time. Returns the time that occurs *n* seconds after *time* if *n* is greater than 0. Returns the time that occurs *n* seconds before *time* if *n* is less than 0. The maximum return value is 23:59:59. If any argument's value is null, RelativeTime returns null.

Examples

This statement returns 19:01:41:

```
RelativeTime(19:01:31, 10)
```

This statement returns 19:01:21:

```
RelativeTime(19:01:31,  - 10)
```

See also

SecondsAfter
RelativeTime method for DataWindows in the *DataWindow Reference*

ReleaseAutomationNativePointer

Description

Releases the pointer to an OLE object that you got with GetAutomationNativePointer.

PocketBuilder	✗
PowerBuilder	✓

Applies to

OLEObject

Syntax

oleobject.**ReleaseAutomationNativePointer** (*pointer*)

Return value

Integer. Returns 0 if it succeeds and -1 if an error occurs.

ReleaseNativePointer

Description

Releases the pointer to an OLE object that you got with GetNativePointer.

PocketBuilder	✗
PowerBuilder	✓

Applies to	OLE controls and OLE custom controls
Syntax	*olename*.**ReleaseNativePointer** (*pointer*)
Return value	Integer. Returns 0 if it succeeds and -1 if an error occurs.

Remove

Removes an object or item at runtime.

To remove	Use
A NotificationBubble object	Syntax 1
An appointment, contact, or task from Pocket Outlook	Syntax 2

Syntax 1 For NotificationBubble objects

Description

Removes a notification bubble and its icon in the notification tray.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✗

Applies to

NotificationBubble object

Syntax

Integer *controlname*.Remove ( )

Argument	Description
controlname	The name of the notification bubble that you want to remove

Return value

Integer. Returns 1 for success, -1 if an error occurs. Typically this is a benign error, because the user has already acknowledged the notification.

Usage

If you do not remove the notification bubble and the user does not acknowledge the notification, the NotificationBubble object could remain in memory and its icon in the notification tray.

Examples

The following example removes a NotificationBubble from a user's system:

```
li_rtn = nb_myBubble.Remove()
```

See also

Icon
SetMessageSink

Syntax 2 For POOM objects

Description

Removes an appointment, contact, or task from Pocket Outlook.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✗

Applies to

POOM objects

Syntax

Integer *objectname*.Remove (*entity*)

Argument	Description
objectname	The name of the POOM object
entity	Entity of type POOMAppointment, POOMContact, or POOMTask that you want to remove

Return value

Integer. Returns 1 for success and one of the following negative values if an error occurs:

-1 Unspecified error

-2 Cannot connect to the repository or a required internal subobject failed to connect to the repository

-3 Cannot log in to the repository

-4 Incorrect input argument

-5 Action cannot be performed

-6 The object identifier (OID) is not in the repository

-7 Feature is not implemented yet

-8 No matching entries found for the criteria

Usage

A user must be logged in to a POOM object to remove an appointment, contact, or task.

Examples

The following example gets the task with the index 3 and removes it from the repository:

```
integer li_rc
POOMTask task

task = g_poom.GetTask( 3 )
li_rc = g_poom.Remove( task )
```

See also

Add

RemoveDirectory

Description

Removes a directory.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

RemoveDirectory (*directoryname*)

Argument	Description
directoryname	String for the name of the directory you want to remove. If you do not specify an absolute path, this function deletes relative to the current working directory.

Return value

Integer. Returns 1 if the function succeeds and -1 if an error occurs.

Usage

The directory must be empty and must not be the current directory for this function to succeed.

Examples

This example removes a subdirectory from the current directory:

```
string  ls_path="my targets"
integer li_filenum

li_filenum = RemoveDirectory ( ls_path )
If li_filename <> 1 then
MessageBox("Remove directory failed", &
   + "Check that the directory exists, is empty, and " &
   + "is not the current directory")
else
MessageBox("Success", "Directory " + ls_path + &
    " deleted")
end if
```

See also

DirectoryExists
GetCurrentDirectory

RemoveRecipient

Description

Removes the specified recipient for the appointment.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✗

Applies to

POOMAppointment

Syntax

Integer *objectname*.RemoveRecipient (*recipient*)

Argument	Description
objectname	The name of the POOMAppointment object
recipient	The POOMRecipient you want to remove from the appointment

Return value

Integer. Returns 1 for success and one of the following negative values if an error occurs:

-1 Unspecified error

-2 Cannot connect to the repository or a required internal subobject failed to connect to the repository

-3 Cannot log in to the repository

-4 Incorrect input argument

-5 Action cannot be performed

-6 The object identifier (OID) is not in the repository

-7 Feature is not implemented yet

-8 No matching entries found for the criteria

See also

AddRecipient
GetRecipients

Repair

Description

Updates the target database with corrections that have been made in the pipeline user object's Error DataWindow.

PocketBuilder	✗
PowerBuilder	✓

Applies to

Pipeline objects

Syntax

pipelineobject.**Repair** (*destinationtrans*)

Return value

Integer. Returns 1 if it succeeds and a negative number if an error occurs.

Replace

Description

Replaces a portion of one string with another.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

Replace (*string1*, *start*, *n*, *string2*)

Argument	Description
string1	The string in which you want to replace characters with *string2*.
start	A long whose value is the number of the first character you want replaced. (The first character in the string is number 1.)
n	A long whose value is the number of characters you want to replace.
string2	The string that will replace characters in *string1*. The number of characters in *string2* can be greater than, equal to, or less than the number of characters you are replacing.

Return value

String. Returns the string with the characters replaced if it succeeds and the empty string if it fails. If any argument's value is null, Replace returns null.

Usage

If the start position is beyond the end of the string, Replace appends *string2* to *string1*. If there are fewer characters after the start position than specified in *n*, Replace replaces all the characters to the right of character *start*.

If *n* is zero, then, in effect, Replace inserts *string2* into *string1*.

Examples

These statements change the value of *Name* from Davis to Dave:

```
string Name
Name = "Davis"
Name = Replace(Name, 4, 2, "e")
```

This statement returns BABY RUTH:

```
Replace("BABE RUTH", 1, 4, "BABY")
```

This statement returns Closed for the Winter:

```
Replace("Closed for Vacation", 12, 8, "the Winter")
```

This statement returns ABZZZZEF:

```
Replace("ABCDEF", 3, 2, "ZZZZ")
```

This statement returns ABZZZZ:

```
Replace("ABCDEF", 3, 50, "ZZZZ")
```

This statement returns ABCDEFZZZZ:

```
Replace("ABCDEF", 50, 3, "ZZZZ")
```

These statements replace all occurrences of red within the string *mystring* with green. The original string is taken from the SingleLineEdit sle_1 and the result becomes the new text of sle_1:

```
long start_pos=1
string old_str, new_str, mystring

mystring = sle_1.Text
old_str = "red"
new_str = "green"

// Find the first occurrence of old_str.
start_pos = Pos(mystring, old_str, start_pos)

// Only enter the loop if you find old_str.
DO WHILE start_pos > 0

    // Replace old_str with new_str.
    mystring = Replace(mystring, start_pos, &
      Len(old_str), new_str)
    // Find the next occurrence of old_str.
    start_pos = Pos(mystring, old_str, &
      start_pos+Len(new_str))
LOOP
```

```
sle_1.Text = mystring
```

See also

Replace method for DataWindows in the *DataWindow Reference*

ReplaceW

Description

Replaces a portion of one string with another.

PocketBuilder	✗
PowerBuilder	✓

Obsolete function
This function is obsolete. It has the same behavior as Replace.

Syntax

ReplaceW (*string1*, *start*, *n*, *string2*)

Return value

String. Returns the string with the characters replaced if it succeeds and the empty string if it fails.

ReplaceText

Description

Replaces selected text in an edit control with a specified string.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

DataWindow, EditMask, MultiLineEdit, SingleLineEdit, RichTextEdit, DropDownListBox, and DropDownPictureListBox controls

Syntax

editname.**ReplaceText** (*string*)

Argument	Description
editname	The name of the control in which you want to replace the selected string. In a DataWindow control, the text is replaced in the edit control over the current row and column.
string	The string that replaces the selected text.

Return value

Long. Returns the number of characters in *string* and -1 if an error occurs. If any argument's value is null, ReplaceText returns null.

Usage

If there is no selection, ReplaceText inserts the replacement text at the cursor position.

In a RichTextEdit control, the selection can include pictures.

Other ways to replace text
To use the contents of the clipboard as the replacement text, call the Paste function, instead of ReplaceText.

To replace text in a string, rather than a control, use the Replace function.

Examples

If the MultiLineEdit mle_Comment contains `Offer Good for 3 Months` and the selected text is `3 Months`, this statement replaces `3 Months` with `60 Days` and returns 7. The resulting value of mle_Comment is `Offer Good for 60 Days`:

```
mle_Comment.ReplaceText("60 Days")
```

If there is no selected text, this statement inserts `"Draft"` at the cursor position in the SingleLineEdit sle_Comment3:

```
sle_Comment3.ReplaceText("Draft")
```

See also

Copy
Cut
Paste

Reset

Clears data from a control or object. The syntax you choose depends on the target object.

For syntax for DataWindows and DataStores see the Reset method for DataWindows in the *DataWindow Reference* or the online Help.

To	Use
Delete all items from a list	Syntax 1
Delete all the data (and optionally the series and categories) from a graph	Syntax 2
Return to the beginning of a trace file	Syntax 3

Syntax 1 For list boxes

Description

Deletes all the items from a list.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

ListBox, DropDownListBox, PictureListBox, and DropDownPictureListBox controls

Syntax

listboxname.**Reset** ( )

Argument	Description
listboxname	The name of the ListBox control from which to delete all items

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs. If *listboxname* is null, Reset returns null. The return value is usually not used.

Examples

This statement deletes all items in the ListBox portion of ddlb_Actions:

```
ddlb_Actions.Reset()
```

See also

DeleteItem

Syntax 2 For graphs

Description

Deletes the data, the categories, or the series from a graph.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Graph controls in windows and user objects and graphs within a DataWindow object with an external data source.

Does not apply to other graphs within DataWindow objects because their data comes directly from the DataWindow.

Syntax

controlname.**Reset** (*graphresettype*)

Argument	Description
controlname	The name of the graph object in which you want to delete all the data values or all series and all data values
graphresettype	A value of the grResetType enumerated datatype specifying whether you want to delete only data values or all series and all data values: • All! — Delete all series, categories, and data in *controlname* • Category! — Delete categories and data in *controlname* • Data! — Delete data in *controlname* • Series! — Delete the series and data in *controlname*

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is null, Reset returns null. The return value is usually not used.

Usage

Use Reset to clear the data in a graph before you add new data.

Examples

This statement deletes the series and data, but leaves the categories, in the graph gr_product_data:

```
gr_product_data.Reset(Series!)
```

See also

AddData
AddSeries

Syntax 3 For trace files

Description

Goes back to the beginning of the trace file so you can begin rereading the file contents.

PocketBuilder on Desktop	✓
PocketBuilder on Pocket PC	✗
PocketBuilder on Smartphone	✗
PowerBuilder	✓

Applies to

TraceFile objects

Syntax

***instancename*.Reset** ( )

Argument	Description
instancename	Instance name of the TraceFile object

Return value

ErrorReturn. Returns one of the following values:

- Success!—The function succeeded
- FileNotOpenError!—The specified trace file has not been opened

Usage

Use this function to return to the start of the open trace file and begin rereading the contents of the file. To use the Reset function, you must have previously opened the trace file with the Open function. You use the Reset and Open functions as well as the other properties and functions provided by the TraceFile object to access the contents of a trace file directly. You use these functions if you want to perform your own analysis of the tracing data instead of using the available modeling objects.

Examples

This example returns execution to the start of the open trace file *ltf_file* so that the file's contents can be reread:

```
TraceFile ltf_file
string ls_filename

ltf_file = CREATE TraceFile
ltf_file.Open(ls_filename)
...
ltf_file.Reset(ls_filename)
...
```

See also

Open
NextActivity
Close

ResetArgElements

Description

Clears the argument list.

PocketBuilder	✗
PowerBuilder	✓

Applies to

Window ActiveX controls

Syntax

activexcontrol.**ResetArgElements** ( )

Return value

Integer. Returns 1 if the function succeeds and -1 if an error occurs.

ResetDataColors

Description

Restores the color of a data point to the default color for its series.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Graph controls in windows and user objects, and graphs in DataWindow controls

Syntax

controlname.**ResetDataColors** ({ *graphcontrol*, } *seriesnumber*, *datapointnumber*)

Argument	Description
controlname	The name of the graph in which you want to reset the color of a data point, or the name of the DataWindow containing the graph
graphcontrol (DataWindow control only)	(Optional) A string whose value is the name of the graph in the DataWindow control in which you want to reset the color
seriesnumber	The number of the series in which you want to reset the color of a data point
datapointnumber	The number of the data point for which you want to reset the color

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is null, ResetDataColors returns null.

Default color for data points
To set the color for a series, use SetSeriesStyle. The color you set for the series is the default color for all data points in the series.

Examples

These statements change the color of data point 10 in the series named *Costs* in the graph gr_product_data to the color for the series:

```
SeriesNbr = gr_product_data.FinSeries("Costs")
gr_product_data.ResetDataColors(SeriesNbr, 10)
```

These statements change the color of data point 10 in the series named *Costs* in the graph gr_comps in the DataWindow control dw_equip to the color for the series:

```
SeriesNbr = dw_equipment.FindSeries("Costs")
dw_equip.ResetDataColors("gr_comps", SeriesNbr, 10)
```

See also

GetDataStyle
SeriesName
GetSeriesStyle
SetDataStyle
SetSeriesStyle

Resize

Description

Resizes an object or control by setting its Width and Height properties and then redraws the object.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Any object, except a child DataWindow

Syntax

objectname.**Resize** (*width*, *height*)

Argument	Description
objectname	The name of the object or control you want to resize
width	The new width in PowerBuilder units
height	The new height in PowerBuilder units

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs or if *objectname* is a minimized or maximized window. If any argument's value is null, Resize returns null.

Usage

You cannot use Resize for a child DataWindow.

Resize does not resize a minimized or maximized sheet or window. If the window is minimized or maximized, Resize returns –1.

Equivalent syntax You can set object's Width and Height properties instead of calling the Resize function. However, the two statements cause PocketBuilder to redraw *objectname* twice; first with the new width, and then with the new width and height.

objectname.Width = *width*

objectname.Height = *height*

The first two statements, although they redraw gb_box1 twice, achieve the same result as the third statement:

```
gb_box1.Width = 100 // These lines resize
gb_box1.Height = 150 // gb_box1 to 100 x 150
gb_box1.Resize(100, 150)// So does this line
```

Examples

This statement changes the Width and Height properties of gb_box1 and redraws gb_box1 with the new properties:

```
gb_box1.Resize(100, 150)
```

This statement doubles the width and height of the picture control p_1:

```
p_1.Resize(p_1.Width*2, p_1.Height*2)
```

Resolve_Initial_References

Description

Uses the CORBA naming service API to obtain the initial naming context for an EAServer component.

This function is used by PowerBuilder clients connecting to EAServer.

PocketBuilder	✗
PowerBuilder	✓

Applies to

JaguarORB objects

Syntax

jaguarorb.**Resolve_Initial_References** (*objstring, object*)

Return value

Long. Returns 0 if it succeeds and a negative number if an error occurs.

RespondRemote

Description

Sends a DDE message indicating whether the command or data received from a remote DDE application was acceptable.

PocketBuilder	✗
PowerBuilder	✓

Syntax

RespondRemote (*boolean*)

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs (for example, the function was called in wrong context). If *boolean* is null, RespondRemote returns null.

Restart

Description

Stops the execution of all scripts, closes all windows (without executing the scripts for the Close events), commits and disconnects from the database, restarts the application, and executes the application-level script for the Open event.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

Restart ( )

Return value

Integer. Returns 1 if it succeeds and -1 if it fails. The return value is usually not used.

Usage

You can use Restart in the application-level script for the Idle event to restart the application after a period of user inactivity, a typical behavior of kiosk applications.

Examples

In the application-level script for the Idle event, this statement restarts the application:

```
Restart()
```

See also

HALT on page 130

ResumeTransaction

Description

Associates the EAServer transaction passed as an argument with the calling thread.

PocketBuilder	✗
PowerBuilder	✓

Applies to

CORBACurrent objects

Syntax

CORBACurrent.**ResumeTransaction** (*handletrans*)

Return value

Integer. Returns 0 if it succeeds and one of the following negative values if an error occurs:

-1 Unknown failure

-2 The transaction referred to by *handletrans* is no longer valid

RetrieveData

Description

Retrieves data from scanner firmware and places it in instance properties of the scanner object.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✗

Applies to

BarcodeScanner objects

Syntax

Integer *scanner*.RetrieveData ( )

Argument	Description
scanner	The scanner object linked to the scanner from which you want to retrieve data

Return value

Integer. Returns 1 for success or one of the following negative values if an error occurs:

- **-1** Unspecified error
- **-2** Supporting DLL not loaded error
- **-3** Initialization error other than DLL not loaded
- **-4** Error in the passed in arguments
- **-5** Something in the object instance is inconsistent
- **-6** Call to the driver failed
- **-7** Error opening the specific scan device
- **-8** Error in the internal buffer allocation
- **-9** Incorrect scan state for the requested action (typically benign)
- **-10** Low level device error

- **-11** Read is already pending (typically benign)
- **-12** Read is cancelled (typically benign)
- **-13** Timeout period expired on the read (typically benign)
- **-14** Error creating the asynchronous read from the message sink
- **-100** Feature not implemented

Usage

After you call RetrieveData, the data from the most recent scan are saved in the BarcodeScanner object's ScannedSymbology and ScannedData properties (data members). You can retrieve the data by assigning these properties to string variables or by displaying them in a text control.

Examples

The following example retrieves data from a single scan:

```
Integer l_iret
l_iret = l_scanner.Open()
l_iret = l_scanner.ScanWait( 30 )
l_iret = l_scanner.RetrieveData()
sle_symbology.text=string(l_scanner.ScannedSymbology)
sle_data.text = l_scanner.ScannedData
```

See also

Open

Reverse

Description

Reverses the order or characters in a string.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

Reverse (*string*)

Argument	Description
string	A string whose characters you want to reorder so that the last character is first and the first character is last

Return value

String. Returns a string with the characters of *string* in reversed order. Returns the empty string if it fails.

Usage

Reverse is useful with the IsArabic and IsHebrew functions, which help you implement right-to-left character display when you are using a version of Windows that supports right-to-left languages.

Examples

Under a a version of Windows that supports right-to-left languages, this statement returns a string with the characters in reverse order from the characters entered in sle_name:

```
string ls_name
ls_name = Reverse(sle_name.Text)
```

See also

IsArabic
IsHebrew

RevertToSelf

Description

Restores the security attributes for a COM object that is running on MTS and impersonating the client.

PocketBuilder	✗
PowerBuilder	✓

Applies to

TransactionServer objects

Syntax

transactionserver.**RevertToSelf** ( )

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs.

RGB

Description

Calculates the long value that represents the color specified by numeric values for the red, green, and blue components of the color.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

RGB (*red*, *green*, *blue*)

Argument	Description
red	The integer value of the red component of the desired color
green	The integer value of the green component of the desired color
blue	The integer value of the blue component of the desired color

Return value

Long. Returns the long that represents the color created by combining the values specified in red, green, and blue. If an error occurs, RGB returns -1. If any argument's value is null, RGB returns null.

Usage

The formula for combining the colors is:

65536 * *Blue*+ 256 * *Green*+ *Red*

Use RGB to obtain the long value required to set the color for text and drawing objects. You can also set an object's color to the long value that represents the color. The RGB function provides an easy way to calculate that value.

About color values
The value of a component of a color is an integer between 0 and 255 that represents the amount of the color that is required to create the color you want. The lower the value, the darker the color; the higher the value, the lighter the color.

To determine the values for the components of a color (known as the RGB values), use the Edit Color Entry window. To access the Edit Color Entry window, select a color in the color bar at the bottom of the workspace and then double-click the selected color when it displays in the first box of the color bar.

The following table lists red, green, and blue values for the 16 standard colors.

Table 10-9: Red, green, and blue color values for use with RGB

Color	Red value	Green value	Blue value
Black	0	0	0
White	255	255	255
Light Gray	192	192	192
Dark Gray	128	128	128
Red	255	0	0
Dark Red	128	0	0
Green	0	255	0
Dark Green	0	128	0
Blue	0	0	255
Dark Blue	0	0	128
Magenta	255	0	255
Dark Magenta	128	0	128
Cyan	0	255	255
Dark Cyan	0	128	128
Yellow	255	255	0
Brown	128	128	0

Examples

This statement returns a long that represents black:

```
RGB(0, 0, 0)
```

This statement returns a long that represents white:

```
RGB(255, 255, 255)
```

These statements set the color properties of the StaticText st_title to be green letters on a dark magenta background:

```
st_title.TextColor = RGB(0, 255, 0)
st_title.BackColor = RGB(128, 0, 128)
```

See also

RGB method for DataWindows in the *DataWindow Reference*

Right

Description

Obtains a specified number of characters from the end of a string.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

Right (*string*, *n*)

Argument	Description
string	The string from which you want characters returned
n	A long whose value is the number of characters you want returned from the right end of *string*

Return value

String. Returns the rightmost *n* characters in *string* if it succeeds and the empty string ("") if an error occurs. If any argument's value is null, Right returns null. If *n* is greater than or equal to the length of the string, Right returns the entire string. It does not add spaces to make the return value's length equal to *n*.

Examples

This statement returns RUTH:

```
Right("BABE RUTH", 4)
```

This statement returns BABE RUTH:

```
Right("BABE RUTH", 75)
```

See also

Left
Mid
Pos
Right method for DataWindows in the *DataWindow Reference*

RightW

Description

Obtains a specified number of characters from the end of a string.

PocketBuilder	✗
PowerBuilder	✓

Obsolete function
This function is obsolete. It has the same behavior as Right in all environments.

Syntax

RightW (*string*, *n*)

Return value

String. Returns the rightmost *n* characters in *string* if it succeeds and the empty string ("") if an error occurs.

RightTrim

Description

Removes spaces from the end of a string.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

RightTrim (*string*)

Argument	Description
string	The string you want returned with trailing blanks deleted

Return value

String. Returns a copy of *string* with trailing blanks deleted if it succeeds and the empty string ("") if an error occurs. If any argument's value is null, RightTrim returns null.

Usage

In SBCS environments, the RightTrim and RightTrimW functiones return the same results. Although you can use the RightTrim function in DBCS environments, it cannot remove double-byte spaces. You must use the RightTrimW function to remove double-byte or mixed single-byte and double-byte spaces.

Examples

This statement returns `RUTH`:

```
RightTrim("RUTH ")
```

See also

LeftTrim
Trim
RightTrim method for DataWindows in the *DataWindow Reference*

RightTrimW

Description

Removes spaces from the end of a string.

PocketBuilder	✗
PowerBuilder	✓

Obsolete function
This function is obsolete. It has the same behavior as RightTrim in all environments.

Syntax

RightTrimW (*string*)

Return value

String. Returns a copy of *string* with trailing blanks deleted if it succeeds and the empty string ("") if an error occurs.

RollbackOnly

Description

Modifies an EAServer transaction associated with a calling thread so that the only possible outcome is to roll back the transaction.

PocketBuilder	✗
PowerBuilder	✓

Applies to

CORBACurrent objects

Syntax

CORBACurrent.**RollbackOnly** ( )

Return value

Integer. Returns 0 if it succeeds and a negative number if an error occurs.

RollbackTransaction

Description

Rolls back the EAServer transaction associated with the calling thread.

PocketBuilder	✗
PowerBuilder	✓

Applies to

CORBACurrent objects

Syntax

CORBACurrent.**RollbackTransaction** ( )

Return value

Integer. Returns 0 if it succeeds and a negative number if an error occurs.

Round

Description

Rounds a number to the specified number of decimal places.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

Round (*x*, *n*)

Argument	Description
x	The number you want to round.
n	The number of decimal places to which you want to round *x*. Valid values are 0 through 18.

Return value

Decimal. Returns *x* rounded to the specified number of decimal places if it succeeds, and null if it fails or if any argument's value is null.

Examples

This statement returns 9.62:

```
Round(9.624, 2)
```

This statement returns 9.63:

```
Round(9.625, 2)
```

This statement returns 9.600:

```
Round(9.6, 3)
```

This statement returns –9.63:

```
Round(-9.625, 2)
```

This statement returns null:

```
Round(-9.625, -1)
```

See also

Ceiling
Int
Truncate
Round method for DataWindows in the *DataWindow Reference*

RoutineList

Description

Provides a list of the routines included in a performance analysis model.

PocketBuilder on Desktop	✓
PocketBuilder on Pocket PC	✗
PocketBuilder on Smartphone	✗
PowerBuilder	✓

Applies to

ProfileClass and Profiling objects

Syntax

***instancename*.RoutineList** (*list*)

Argument	Description
instancename	Instance name of the ProfileClass or Profiling object.
list	An unbounded array variable of datatype ProfileRoutine in which RoutineList stores a ProfileRoutine object for each routine that exists in the model within a class. This argument is passed by reference.

Return value

ErrorReturn. Returns one of the following values:

- Success!—The function succeeded
- ModelNotExistsError! —No model exists

Usage

Use this function to extract a list of the routines included in the performance analysis model in a particular class. You must have previously created the performance analysis model from a trace file using the BuildModel function. Each routine is defined as a ProfileRoutine object and provides the time spent in the routine, any called routines, the number of times each routine was called, and the class to which the routine belongs. The routines are listed in no particular order.

Object creation and destruction for a class are each indicated by a routine in this list as well as by embedded SQL statements.

Examples

This example lists the routines included in each class found in a performance analysis model:

```
Long ll_cnt
ProfileCall lproc_call[]

lpro_model.BuildModel()
lpro_model.RoutineList(iprort_list)
...
```

See also ClassList

Run

Description

Runs the specified application program.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

Run (*string* {, *windowstate* })

Argument	Description
string	A string whose value is the file name of the program you want to execute. Optionally, *string* can contain one or more parameters for the program.
windowstate (optional)	A value of the WindowState enumerated datatype indicating the state in which you want to run the program: • Maximized! — Maximized; enlarge the program window to its maximum size when it starts • Minimized! — Minimized; shrink the program window to an icon when it starts • Normal! — (Default) Run the program window in its normal size

Return value

Integer. Returns 1 if it is successful and -1 if an error occurs. If any argument's value is null, Run returns null.

Usage

You can use Run for any program that you can run from the operating system. If you do not specify parameters, Run opens the application and displays the first application window. If you specify *windowstate*, the application window is displayed in the specified state.

If you specify parameters, the application determines the meaning of those parameters. A typical use is to identify a data file to be opened when the program is executed. If you are running another PocketBuilder application, that application can call the CommandParm function to retrieve the parameters and process them as it sees fit.

If the file extension is omitted from the file name, PocketBuilder assumes the extension is *.EXE*. To run a program with another extension (for example, *.BAT*, *.COM*, or *.PIF*), you must specify the extension.

Examples

This statement runs the Microsoft Windows Clock accessory application in its normal size:

```
Run("Clock")
```

This statement runs the Microsoft Windows Clock accessory application minimized:

```
Run("Clock", Minimized!)
```

This statement runs the program *WINNER.COM* on the C drive in a maximized state. The parameter passed to *WINNER.COM* opens the file *EMPLOYEE.INF*:

```
Run("C:\WINNER.COM EMPLOYEE.INF", Maximized!)
```

This example runs the DOS batch file *MYBATCH.BAT* and passes the parameter `TEST` to the batch file. In the batch file, you include percent substitution characters in the commands to indicate where the parameter is used:

```
Run("MYBATCH.BAT TEST")
```

In the batch file the following statement renames *FILE1* to *TEST*:

```
RENAME c:\PB\FILE1 %1
```

RunSync

Description

Launches the MobiLink *DBMLSync.exe* process and waits for it to complete.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✗

Syntax

RunSync (*exename*, *publication*, *mluser*, *mlpassword*, *connectstring*, *otherparams*, *windowclassname*)

Argument	Description
exename	A read-only string (passed by reference) that defines the location of the MobiLink synchronization client *DBMLSync.exe* file
publication	A read-only string for the name of the publication or the names of the publications you want to synchronize on the remote and consolidated databases
mluser	A read-only string for the MobiLink user name
mlpassword	A read-only string for the MobiLink password
connectstring	A read-only string for the remote database connection parameters or for the location of the DSN file containing the connection parameters
otherparams	A read-only string for any logging, command line, and extended options that you want to use with the DBMLSync process
windowclassname	A read-only string for the handle to the window that you want to have receive messages from the dbmlsync process

Return value

Long. Returns 1 for success and -1 if another instance of the synchronization client is currently running. *DBMLSync.exe* process errors return positive error codes.

Usage

You can use the RunSync function to start the DBMLSync process. If you use the MobiLink Synchronization for ASA wizard, it is not necessary to make this call explicitly. This is because the uf_runsync function of the wizard-generated user object makes an equivalent call to pb_run_dbmlsync in the PocketBuilder VM, using parameters based on instance variables generated by the wizard, selected by the end user, or obtained from the device registry.

The MobiLink Synchronization Reference describes permitted values for the *otherparams* argument. These values are also selectable in the MobiLink Synchronization for ASA wizard.

See also

CancelSync

Save

Description

Saves an OLE object in an OLE control or an OLE storage object.

PocketBuilder	✗
PowerBuilder	✓

Syntax

oleobject.**Save** ( )

Return value

Integer. Returns 0 if it succeeds and a negative number if an error occurs.

SaveAs

Saves the contents of a DataWindow, DataStore, graph, OLE control, or OLE storage in a file. The syntax you use depends on the type of object you want to save.

For DataWindow and DataStore syntax, see the SaveAs method for DataWindows in the *DataWindow Reference* or the online Help.

To	Use
Save the data in a graph	Syntax 1
Save the OLE object in an OLE control to a storage file	Syntax 2
Save the OLE object in an OLE control to a storage object in memory	Syntax 3
Save an OLE storage and any controls that have opened that storage in a file	Syntax 4
Save an OLE storage object in another OLE storage object	Syntax 5

Syntax 1 For graph objects

Description

Saves the data in a graph in the format you specify.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Graph controls in windows and user objects, and graphs in DataWindow controls and DataStores

Syntax

controlname.**SaveAs** ({ *filename*, } { *graphcontrol*, *saveastype*, *colheading* })

Argument	Description
controlname	The name of the graph control whose contents you want to save or the name of the DataWindow DataStore containing the graph.
filename (optional)	A string whose value is the name of the file in which you want to save the data in the graph. If you omit *filename* or specify an empty string (""), PocketBuilder prompts the user for a file name.
graphcontrol (DataWindow control only) (optional)	A string whose value is the name of the graph in the DataWindow control or DataStore whose contents you want to save.
saveastype (optional)	A value of the SaveAsType enumerated datatype specifying the format in which to save the data represented in the graph. Values are: • CSV! — Comma-separated values • DIF! — Data Interchange Format • Excel! — Microsoft Excel format • SQLInsert! — SQL syntax • Text! — (Default) Tab-separated columns with a return at the end of each row
colheading (optional)	A boolean value indicating whether you want column headings with the saved data. The default value is true. *Colheading* is ignored for DIF files; column headings are always saved.

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is null, SaveAs returns null.

Usage

You must use zero or three arguments. If you do not specify any arguments for SaveAs, PocketBuilder displays the Save As dialog box, letting the user specify the format of the saved data.

Regional settings

If you use date formats in your graph, you must verify that yyyy is the Short Date Style for year in the Regional Settings of the user's Control Panel. Your program can check this with the RegistryGet function.

If the setting is not correct, you can ask the user to change it manually or to have the application change it (by calling the RegistrySet function). The user may need to reboot after the setting is changed.

Examples

This statement saves the contents of the graph gr_History. The file and format information are not specified, so PocketBuilder prompts for the file name and save the graph as tab-delimited text:

```
gr_History.SaveAs()
```

This statement saves the contents of gr_History to the file *\WINDOWS\HR\EMPLOYEE.HIS*. The format is CSV without column headings:

```
gr_History.SaveAs("\WINDOWS\HR\EMPLOYEE.HIS",CSV!,&
       FALSE)
```

This statement saves the contents of gr_computers in the DataWindow control dw_equipmt to the file *G:\INVENTORY\SALES.XLS*. The format is Excel with column headings:

```
dw_equipmt.SaveAs("gr_computers", &
       "G:\INVENTORY\SALES.XLS", Excel!, TRUE)
```

See also

Print

Syntax 2 For saving an OLE control to a file

Description

Saves the object in an OLE control in a storage file.

PocketBuilder	✗
PowerBuilder	✓

Applies to

OLE controls

Syntax

olecontrol.**SaveAs** (*OLEtargetfile*)

Return value

Integer. Returns 0 if it succeeds and a negative number if an error occurs.

Syntax 3 For saving an OLE control to an OLE storage

Description

Saves the object in an OLE control to an OLE storage object in memory.

PocketBuilder	✗
PowerBuilder	✓

Applies to

OLE controls

Syntax

olecontrol.**SaveAs** (*targetstorage*, *substoragename*)

Return value — Integer. Returns 0 if it succeeds and a negative number if an error occurs.

Syntax 4 — For saving an OLE storage object to a file

Description — Saves an OLE storage object to a file. If OLE controls have opened the OLE storage object, this syntax of SaveAs puts them in a saved state too.

PocketBuilder	✗
PowerBuilder	✓

Applies to — OLE storage objects

Syntax — *olestorage*.**SaveAs** (*OLEtargetfile*)

Return value — Integer. Returns 0 if it succeeds and a negative number if an error occurs.

Syntax 5 — For saving an OLE storage object in another OLE storage

Description — Saves an OLE storage object to another OLE storage object variable in memory.

PocketBuilder	✗
PowerBuilder	✓

Applies to — OLE storage objects

Syntax — *olestorage*.**SaveAs** (*substoragename*, *targetstorage*)

Return value — Integer. Returns 0 if it succeeds and a negative number if an error occurs.

SaveDocument

Description — Saves the contents of a RichTextEdit control in a file. You can specify either rich-text format (RTF) or ASCII text format for the file.

PocketBuilder	✗
PowerBuilder	✓

Applies to — RichTextEdit controls

Syntax — *rtename*.**SaveDocument** (*filename* {, *filetype* })

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs.

ScanAbort

Description

Aborts any outstanding scan requests.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✗

Applies to

BarcodeScanner objects

Syntax

Integer *scanner.*ScanAbort ( )

Argument	Description
scanner	The scanner object for which you abort a scan

Return value

Integer. Returns 1 for success or one of the following negative values if an error occurs:

- **-1** Unspecified error
- **-2** Supporting DLL not loaded error
- **-3** Initialization error other than DLL not loaded
- **-4** Error in the passed in arguments
- **-5** Something in the object instance is inconsistent
- **-6** Call to the driver failed
- **-7** Error opening the specific scan device
- **-8** Error in the internal buffer allocation
- **-10** Low level device error
- **-100** Feature not implemented

Examples

The following example aborts the scan operation for the l_scanner bar code scanner:

```
li_rtn = l_scanner.ScanAbort()
```

See also

Flush
ScanWait

ScanCapture

Description

Starts a synchronous scan.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✗

Applies to

BiometricScanner objects

Syntax

Integer *scanner.*ScanCapture (*timeout*, *biometricpurpose*)

Argument	Description
scanner	The scanner object associated with the device you want to use to complete a scan
timeout	Integer value for the period in seconds after which a scan will return
biometricpurpose	Enumerated value for the type of scan. Values are: • purposeaudit! • purposeenroll! • purposeenrollforidentificationonly! • purposeenrollforverificationonly! • purposeidentify! • purposeverify!

Return value

Integer. Returns 1 for success or one of the following negative values if an error occurs:

Error	Description
-1	General error
-2	Supporting DLL not loaded error
-3	Initialization error other than DLL not loaded
-4	Error in the passed in arguments
-5	Something in the object instance is inconsistent
-6	Call to the driver failed
-7	Error opening the specific scan device
-8	Error in the internal buffer allocation
-9	Incorrect scan state for the requested action
-10	Low level device error
-11	Read is already pending
-12	Read is cancelled

Error	Description
-13	Timeout period expired on the read
-14	Verification error
-15	Signature error
-16	Data handle error
-17	Inconsistent purpose error
-18	Unsupported purpose error
-19	Record not found error
-20	Scan capture error
-21, -22, -23, -24	Internal scanner error
-25	No image available error
-100	Feature not implemented

Usage

Calling ScanCapture starts a synchronous scan operation. The scan returns only when a value has been scanned or the timeout period has expired.

Examples

The following scenario scans a fingerprint and compares it to stored data for verification purposes:

```
Integer l_iret
Integer l_iQuality
Blob l_blbMinutiae, l_blbMinutiaeFromScan

BiometricScanner l_scanner
l_scanner = CREATE HPBiometricScanner
l_iret = l_scanner.Open()
l_iret = l_scanner.ScanCapture(30, &
   EnrollForVerification!)
sle_quality.text = string(l_scanner.ScannedQuality())
l_iret = l_scanner.ScannedMinutiae(l_blbMinutiae)

l_iret = l_scanner.VerifyMatch(l_blbMinutiaeFromScan, &
   l_blbMinutiae)
DESTROY l_scanner
```

See also

ScannedQuality
VerifyMatch

ScannedBitmap

Description

Retrieves a Windows bitmap image from the most recent scan.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✗

Applies to

BiometricScanner objects

Syntax

Integer *scanner*.ScannedBitmap (*data* {, *height*, *width*})

Argument	Description
scanner	The scanner object associated with the device you want to use to complete a scan
data	Blob value for the image passed by reference
height	Integer value in pixels for the height of the scanned image (not implemented by the HPBiometricScanner object)
width	Integer value in pixels for the width of the scanned image (not implemented by the HPBiometricScanner object)

Return value

Integer. Returns 1 for success or a negative value if an error occurs. For a list of possible errors and their definitions, see ScanCapture on page 868.

Usage

The ScannedBitmap function provides visual feedback about the actual fingerprint scanned. This function is not used in verification calculations—it has no algorithmic purpose.

For HPBiometricScanner objects, use the syntax with the *data* argument only.

Examples

The following example passes the scanned image to a local variable with a blob datatype, and uses the default image size:

```
li_rtn = l_bioscanner.ScannedBitmap(lb_mydata)
```

See also

ScanCapture
ScannedMinutiae
ScannedQuality
SetPicture

ScannedMinutiae

Description

Retrieves the encoded minutiae buffer from the most recent scan.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✗

Applies to

BiometricScanner objects

Syntax

Integer *scanner*.ScannedMinutiae (*data*)

Argument	Description
scanner	The scanner object associated with the device you want to use to complete a scan
data	Blob value for the encoded data passed by reference

Return value

Integer. Returns 1 for success or a negative value if an error occurs. For a list of possible errors and their definitions, see ScanCapture on page 868.

Usage

The value passed in the *data* argument is either stored in a database or used for verification. It represents, in an abstract manner, the structure of a fingerprint (its loops and whorls) for use by verification algorithms.

Examples

The following example passes the scanned data to a local variable with a blob datatype:

```
li_rtn = l_bioscanner.ScannedMinutiae(lb_mydata)
```

See also

ScanCapture
ScannedBitmap
ScannedQuality

ScannedQuality

Description

Retrieves the quality rating from the most recent scan.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✗

Applies to

BiometricScanner objects

Syntax

Integer *scanner.*ScannedQuality ( )

Argument	Description
scanner	The scanner object associated with the device you want to use to complete a scan

Return value

Integer. Returns one of the following values for the quality of the most recent scan:

- **0** Poor quality
- **1** Acceptable quality
- **2** Good quality

Usage

The quality value is generated during a scan by an algorithm set on the scanner.

Examples

The following example passes the quality of the most recent scan to a local variable:

```
li_rtn = l_bioscanner.ScannedQuality( )
```

See also

ScanCapture
ScannedBitmap
ScannedMinutiae

ScanNoWait

Description

Starts a scan operation, but returns immediately, permitting continuous scans.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✗

Applies to

BarcodeScanner objects

Syntax

Integer *scanner.*ScanNoWait ( )

Argument	Description
scanner	The scanner object for which you want to set up continuous scanning

Return value

Integer. Returns 1 for success or one of the following negative values if an error occurs:

- **-1** Unspecified error

- **-2** Supporting DLL not loaded error
- **-3** Initialization error other than DLL not loaded
- **-4** Error in the passed in arguments
- **-5** Something in the object instance is inconsistent
- **-6** Call to the driver failed
- **-7** Error opening the specific scan device
- **-8** Error in the internal buffer allocation
- **-9** Incorrect scan state for the requested action (typically benign)
- **-10** Low level device error
- **-11** Read is already pending (typically benign)
- **-12** Read is cancelled (typically benign)
- **-14** Error creating the asynchronous read from the message sink
- **-100** Feature not implemented

Usage

The ScanNoWait function is used to set an asynchronous scan operation. In a typical implementation, the ScanNoWait call is made in the ScanTriggered event, which leads to continuous (asynchronous) scan readings.

Examples

The following example in the script for the ScanTriggered event sets the scanner for continuous operation:

```
li_rtn = l_scanner.ScanNoWait()
```

See also

ScanWait

ScanWait

Description

Starts a scan operation that returns only after the scan data has been read or the scan timeout period has expired.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✗

Applies to

BarcodeScanner objects

Syntax

Integer *scanner.*ScanWait (*timeout*)

Argument	Description
scanner	The scanner object for which you want to perform the scan
timeout	Integer value for the time period in seconds after which scan data is returned

Return value

Integer. Returns 1 for success or one of the following negative values if an error occurs:

- **-1** Unspecified error
- **-2** Supporting DLL not loaded error
- **-3** Initialization error other than DLL not loaded
- **-4** Error in the passed in arguments
- **-5** Something in the object instance is inconsistent
- **-6** Call to the driver failed
- **-7** Error opening the specific scan device
- **-8** Error in the internal buffer allocation
- **-9** Incorrect scan state for the requested action (typically benign)
- **-10** Low level device error
- **-11** Read is already pending (typically benign)
- **-12** Read is cancelled (typically benign)
- **-13** Timeout period expired on the read (typically benign)
- **-100** Feature not implemented

Usage

For a synchronous scan operation, the scanner object returns data either immediately upon expiration of the timeout period set in the ScanWait call, or when a new scan is started.

Socket bar code scanner
ScanWait is not supported by the Socket bar code scanner.

Examples

The following example starts a scan and reads the data:

```
li_rtn = l_scanner.ScanWait(30)
li_rtn = l_scanner.RetrieveData( )
ls_value = l_scanner.ScannedData
```

See also

RetrieveData
ScanAbort

ScanNoWait

Scroll

Description

Scrolls a multiline edit control or the edit control of a DataWindow a specified number of lines up or down.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

DataWindow, MultiLineEdit, and RichTextEdit controls

Syntax

editname.**Scroll** (*number*)

Argument	Description
editname	The name of the DataWindow, RichTextEdit, or MultiLineEdit in which you want to scroll up or down. If *editname* is a DataWindow, then Scroll affects its edit control.
number	A long specifying the direction and number of lines you want to scroll. To scroll down, use a positive long value. To scroll up, use a negative long value.

Return value

Long. Scroll returns the line number of the first visible line in *editname* if it succeeds. Scroll returns -1 if an error occurs. If any argument's value is null, Scroll returns null.

Usage

If the number of lines left in the list is less than the number of lines that you want to scroll, then Scroll scrolls to the beginning or end, depending on the direction specified.

Examples

This statement scrolls mle_Employee down 4 lines:

```
mle_Employee.Scroll(4)
```

This statement scrolls mle_Employee up 4 lines:

```
mle_Employee.Scroll(-4)
```

See also

ScrollNextPage
ScrollNextRow
ScrollPriorPage
ScrollPriorRow
ScrollToRow

ScrollNextPage

Description	Scrolls to the next page of the document in a RichTextEdit control or RichTextEdit DataWindow.

PocketBuilder	✗
PowerBuilder	✓

For DataWindow syntax, see the ScrollNextPage method for DataWindows in the *DataWindow Reference* or the online Help.

Applies to	RichTextEdit controls
Syntax	*rtename*.**ScrollNextPage** ( )
Return value	Integer. Returns 1 if it succeeds and -1 if an error occurs.

ScrollNextRow

Description	Scrolls to the next instance of the document in a RichTextEdit control or RichTextEdit DataWindow. A RichTextEdit control has multiple instances of its document when it shares data with a DataWindow. The next instance of the document is associated with the next row in the DataWindow.

PocketBuilder	✗
PowerBuilder	✓

For syntax specific to DataWindow controls and child DataWindows, see the ScrollNextRow method for DataWindows in the *DataWindow Reference* or the online Help.

Applies to	DataWindow and RichTextEdit controls
Syntax	*rtename*.**ScrollNextRow** ( )
Return value	Integer. Returns 1 if it succeeds and -1 if an error occurs.

ScrollPriorPage

Description	Scrolls to the prior page of the document in a RichTextEdit control or RichTextEdit DataWindow.

PocketBuilder	✗
PowerBuilder	✓

For syntax specific to DataWindow controls and child DataWindows, see the ScrollPriorPage method for DataWindows in the *DataWindow Reference* or the online Help.

Applies to	DataWindow and RichTextEdit controls
Syntax	*rtename*.**ScrollPriorPage** ( )
Return value	Integer. Returns 1 if it succeeds and -1 if an error occurs.

ScrollPriorRow

Description	Scrolls to the prior instance of the document in a RichTextEdit control or RichTextEdit DataWindow. A RichTextEdit control has multiple instances of its document when it shares data with a DataWindow. The next instance of the document is associated with the next row in the DataWindow.

PocketBuilder	✗
PowerBuilder	✓

For syntax specific to DataWindow controls and child DataWindows, see the ScrollPriorRow method for DataWindows in the *DataWindow Reference* or the online Help.

Applies to	DataWindow and RichTextEdit controls
Syntax	*rtename*.**ScrollPriorRow** ( )
Return value	Integer. Returns 1 if it succeeds and -1 if an error occurs.

ScrollToRow

Description

Scrolls to the document instance associated with the specified row when the RichTextEdit controls shares data with a DataWindow.

PocketBuilder	✗
PowerBuilder	✓

For syntax specific to DataWindow controls and child DataWindows, see the ScrollToRow method for DataWindows in the *DataWindow Reference* or the online Help.

Applies to

RichTextEdit controls

Syntax

rtename.**ScrollToRow** (*row*)

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs.

Second

Description

Obtains the number of seconds in the seconds portion of a time value.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

Second (*time*)

Argument	Description
time	The time value from which you want the seconds

Return value

Integer. Returns the seconds portion of *time* (00 to 59). If *time* is null, Second returns null.

Examples

This statement returns 31:

```
Second(19:01:31)
```

See also

Hour
Minute
Second method for DataWindows in the *DataWindow Reference*

SecondsAfter

Description

Determines the number of seconds one time occurs after another.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

SecondsAfter (*time1*, *time2*)

Argument	Description
time1	A time value that is the start time of the interval being measured
time2	A time value that is the end time of the interval

Return value

Long. Returns the number of seconds *time2* occurs after *time1*. If *time2* occurs before *time1*, SecondsAfter returns a negative number. If any argument's value is null, SecondsAfter returns null.

Examples

This statement returns 15:

```
SecondsAfter(21:15:30, 21:15:45)
```

This statement returns -15:

```
SecondsAfter(21:15:45, 21:15:30)
```

This statement returns 0:

```
SecondsAfter(21:15:45, 21:15:45)
```

If you declare *start_time* and *end_time* time variables and assign 19:02:16 to *start_time* and 19:02:28 to end_time as shown below:

```
time start_time, end_time
start_time = 19:02:16
end_time = 19:02:28
```

then each of these statements returns 12:

```
SecondsAfter(start_time, end_time)
SecondsAfter(19:02:16, end_time)
SecondsAfter(start_time, 19:02:28)
SecondsAfter(19:02:16, 19:02:28)
```

See also

DaysAfter
RelativeDate
RelativeTime
SecondsAfter method for DataWindows in the *DataWindow Reference*

Seek

Moves the pointer to the specified position in an OLEStream object or in a file that you open using the FileDirect object.

To move the pointer in	Use
an OLEStream object	Syntax 1
a file opened by the FileDirect object	Syntax 2

Syntax 1 For OLEStream objects

Description

Moves the read/write pointer to the specified position in an OLE stream object. The pointer is the position in the stream at which the next read or write begins.

PocketBuilder	✗
PowerBuilder	✓

Applies to

OLEStream objects

Syntax

olestream.**Seek** (*position* {, *origin* })

Return value

Integer. Returns 0 if it succeeds and a negative number if an error occurs.

Syntax 2 For FileDirect objects

Description

Moves the pointer in a file that you open with the FileDirect object.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✗

Applies to

FileDirect objects

Syntax

instancename.Seek (*distanceToMove*, *mode*)

Argument	Description
instancename	Instance name of the FileDirect object
distanceToMove	Long for the number of bytes by which you want to move the file pointer. You move the pointer from the position specified by the *mode* argument

Argument	Description
mode	Enumerated value of type seektype. Values can be: • **frombeginning!** Move the pointer from the file beginning • **fromcurrent!** Move the pointer from the current position • **fromend!** Move the pointer from the end of the file

Return value

Integer. Returns 1 for success and a negative number for an error.

Usage

Use the Seek function to place the file pointer at a specified position before you begin to read from or write to the file.

Examples

The following script moves the file pointer 100 bytes from the file end before the Read function is called:

```
li_ret = nvo_FileDirect.seek ( 100, fromend!)
li_ret = nvo_FileDirect.read ( lb_data, 100, li_amount)
```

See also

Open
Read
SetEndOfFile

SelectedColumn

Description

Obtains the number of the character column just after the insertion point in a RichTextEdit control.

PocketBuilder	✗
PowerBuilder	✓

Applies to

RichTextEdit controls

Syntax

rtename.**SelectedColumn** ( )

Return value

Integer. Returns the number of the character before the insertion point in *rtename*. If an error occurs, SelectedColumn returns -1.

SelectedIndex

Description

Obtains the number of the selected item in a ListBox or ListView control.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

ListBox and ListView controls

Syntax

listcontrolname.**SelectedIndex** ( )

Argument	**Description**
listcontrolname	The name of the ListBox or ListView control in which you want to locate the selected item

Return value

Integer. Returns the index of the selected item in *listcontrolname*. If more than one item is selected, SelectedIndex returns the index of the first selected item. If there are no selected items or an error occurs, SelectedIndex returns -1. If *listcontrolname* is null, SelectedIndex returns null.

Usage

SelectedIndex and SelectedItem are meant for lists that allow a single selection only (when the MultiSelect property for the control is false).

When the MultiSelect property is true, SelectedIndex gets the index of the first selected item only. Use the State function, instead of SelectedIndex, to check each item in the list and find out if it is selected. Use the Text function to get the text of any item in the list.

Examples

If item 5 in *lb_actions* is selected, then this example sets *li_Index* to 5:

```
integer li_Index
li_Index = lb_actions.SelectedIndex()
```

These statements open the window w_emp if item 5 in *lb_actions* is selected:

```
integer li_X
li_X = lb_actions.SelectedIndex()
If li_X = 5 then Open(w_emp)
```

See also

SelectedItem

SelectedItem

Description

Obtains the text of the selected item in a ListBox control.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

ListBox and PictureListBox controls

Syntax

listboxname.**SelectedItem** ( )

Argument	Description
listboxname	The name of the ListBox or PictureListBox in which you want the text of the currently selected item

Return value

String. Returns the text of the selected item in *listboxname*. Returns the empty string ("") if no items are selected. If *listboxname* is null, SelectedItem returns null.

Usage

SelectedIndex and SelectedItem are meant for lists that allow a single selection only (when the MultiSelect property for the control is false).

When the MultiSelect property is true, SelectedItem gets the text of the first selected item only. Use the State function, instead of SelectedItem, to check each item in the list and find out if it is selected. Use the Text function to get the text of any item in the list.

Examples

If the text of the selected item in the ListBox lb_shortcuts is F1, then this example sets *ls_item* to F1:

```
string ls_Item
ls_Item = lb_Shortcuts.SelectedItem()
```

See also

SelectedIndex
State

SelectedLength

Description
Determines the total number of characters in the selected text in an editable control, including spaces and line endings.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to
DataWindow, EditMask, MultiLineEdit, SingleLineEdit, RichTextEdit, DropDownListBox, and DropDownPictureListBox controls

Syntax
editname.**SelectedLength** ( )

Argument	Description
editname	The name of the control in which you want the length of the selected text. For a DataWindow, it reports the length of the selected text in the edit control over the current row and column.

Return value
Long. Returns the length of the selected text in *editname*. If no text is selected, SelectedLength returns 0. If an error occurs, it returns -1. If *editname* is null, SelectedLength returns null.

Usage
The characters that make up a line ending, produced by typing Ctrl+Enter or Enter, is different on different platforms. On Windows, it is a carriage return plus a line feed and equals two characters when calculating the length. On other platforms, a line ending is a single character. A line that has wrapped has no line-ending character. For DropDownListBox and DropDownPictureListBox controls, SelectedLength returns -1 if the control's AllowEdit property is set to false.

Focus and the selection in a drop-down list
When a DropDownListBox or DropDownPictureListBox loses focus, the selected text is no longer selected.

Examples
If the selected text in the MultiLineEdit mle_Contact is John Smith, then this example sets *li_length* to 10:

```
integer li_length
li_length = mle_Contact.SelectedLength()
```

See also
LineLength
SelectedItem
SelectedLine

SelectedPage
SelectedStart
TextLine

SelectedLine

Description

Obtains the number of the line that contains the insertion point in an editable control.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

DataWindow, MultiLineEdit, and RichTextEdit controls

Syntax

editname.**SelectedLine** ( )

Argument	Description
editname	The name of the DataWindow, MultiLineEdit, or RichTextEdit in which you want the number of the line containing the insertion point. For a DataWindow, it reports the line number in the edit control over the current row and column.

Return value

Long. Returns the number of the line containing the insertion point in *editname*. If an error occurs, SelectedLine returns -1. If *editname* is null, SelectedLine returns null.

Usage

For EditMask controls, SelectedLine compiles but always returns 1.

The insertion point can be at the beginning or end of the selection. Therefore, SelectedLine can return the first or last selected line, depending on the position of the insertion point.

Examples

If the insertion point is positioned anywhere in line 5 of the MultiLineEdit mle_Contact, the following example sets *li_SL* to 5:

```
integer li_SL
li_SL = mle_Contact.SelectedLine()
```

In this example, the line the user selects in the MultiLineEdit mle_winselect determines which window to open:

```
integer li_SL
li_SL = mle_winselect.SelectedLine()
```

```
IF li_SL = 1 THEN
      Open(w_emp_data)
ELSEIF li_SL = 2 THEN
      Open(w_dept_data)
END IF
```

See also

LineLength
Position
SelectedColumn
SelectedPage
SelectedText
TextLine

SelectedPage

Description

Obtains the number of the current page in a RichTextEdit control.

PocketBuilder	✗
PowerBuilder	✓

Applies to

RichTextEdit controls

Syntax

rtename.**SelectedPage** ( )

Return value

Integer. Returns the number of the current page in *rtename*. If an error occurs, SelectedPage returns -1.

SelectedStart

Description

Reports the position of the first selected character in an editable control.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

DataWindow, EditMask, MultiLineEdit, SingleLineEdit, RichTextEdit, DropDownListBox, and DropDownPictureListBox controls

Syntax

editname.**SelectedStart** ( )

Argument	Description
editname	The name of the control in which you want to determine the starting position of selected text. For a DataWindow, it reports the starting position in the edit control over the current row and column.

Return value

Long. Returns the starting position of the selected text in *editname*. If no text is selected, SelectedStart returns the position of the character immediately following the insertion point. If an error occurs, SelectedStart returns -1. If *editname* is null, SelectedStart returns null.

Usage

For all controls except RichTextEdit, SelectedStart counts from the start of the text and includes spaces and line endings.

For RichTextEdit controls, SelectedStart counts from the start of the line on which the selection begins. The start is at the opposite end of the selection from the insertion point. For example, if the user dragged back to make the selection, the start of the selection is at the end of the highlighted text and the insertion point is before the start. Use the Position function to get information about the start *and* end of the selection.

Focus and the selection in a drop-down list
When a DropDownListBox or DropDownPictureListBox loses focus, the selected text is no longer selected.

Examples

If the MultiLineEdit mle_Comment contains `Closed for Vacation July 3 to July 10`, and `Vacation` is selected, then this example sets *li_Start* to 12 (the position of the first character in `Vacation`):

```
integer li_Start
li_Start = mle_Comment.SelectedStart()
```

See also

Position
SelectedLine
SelectedPage

SelectedText

Description

Obtains the selected text in an editable control.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

DataWindow, EditMask, MultiLineEdit, SingleLineEdit, RichTextEdit, DropDownListBox, and DropDownPictureListBox controls

Syntax

editname.**SelectedText** ( )

Argument	Description
editname	The name of the control from which you want the selected text. For a DropDownListBox or DropDownPictureListBox, the AllowEdit property must be true. For a DataWindow, it reports the selected text in the edit control over the current row and column.

Return value

String. Returns the selected text in *editname*. If there is no selected text or if an error occurs, SelectedText returns the empty string (""). If *editname* is null, SelectedText returns null.

Usage

In a RichTextEdit control, any pictures in the selection are ignored. If the selection contains input fields, the names of the input fields, enclosed in brackets, become part of the string SelectedText returns. The contents of the input fields are not returned.

For example, when the salutation of a letter is selected, SelectedText might return:

```
Dear {title} {lastname}:
```

Focus and the selection in a drop-down list
When a DropDownListBox or DropDownPictureListBox loses focus, the selected text is no longer selected.

Examples

If the text in the MultiLineEdit mle_Contact is `James B. Smith` and `James B.` is selected, these statements set the value of *emp_fname* to `James B`:

```
string ls_emp_fname
ls_emp_fname = mle_Contact.SelectedText()
```

If the selected text in the edit portion of the DropDownListBox ddlb_Location is Maine, these statements display the ListBox lb_LBMaine:

```
string ls_Loc
ls_Loc = ddlb_Location.SelectedText()
IF ls_Loc = "Maine" THEN
      lb_LBMaine.Show()
ELSE
      ...
END IF
```

See also SelectText

SelectionRange

Description Highlights a range of contiguous values in a trackbar control. The range you select is highlighted in the trackbar channel, with an arrow at each end of the range.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✓

Applies to Trackbar controls

Syntax *control*.**SelectionRange** (*startpos*, *endpos*)

Argument	Description
control	The name of the trackbar control
startpos	An integer that specifies the starting position of the range
endpos	An integer that specifies the ending position of the range

Return value Integer. Returns 1 if it succeeds and -1 if an error occurs.

Usage Use this function to indicate a range of preferred values.

In a scheduling application, the selection range could indicate a block of time that is unavailable. Setting a selection range does not prevent the user from selecting a value either inside or outside the range.

Examples This statement highlights the trackbar values between 30 and 70:

```
HTrackBar.SelectionRange( 30, 70 )
```

See also

HTrackBar in *Objects and Controls*
VTrackBar in *Objects and Controls*

SelectItem

Finds and highlights an item in a ListBox, DropDownListBox, or TreeView control.

To select an item	Use
In a ListBox control when you know the text of the item, but not its position	Syntax 1
In a ListBox control when you know the position of the item in the control's list, or to clear the current selection	Syntax 2
In a TreeView control	Syntax 3

Syntax 1 When you know the text of an item

Description

Finds and highlights an item in a ListBox when you can specify some or all of the text of the item.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

ListBox, DropDownListBox, PictureListBox, and DropDownPictureListBox controls

Syntax

listboxname.**SelectItem** (*item*, *index*)

Argument	Description
listboxname	The name of the ListBox control in which you want to select a line
item	A string whose value is the starting text of the item you want to select
index	The number of the item after which you want to begin the search

Return value

Integer. Returns the index number of the selected item. If no match is found, SelectItem returns 0; it returns -1 if an error occurs. If any argument's value is null, SelectItem returns null.

Usage

SelectItem begins searching for the desired item after the item identified by *index*. To match, the item must start with the specified text; however, the text in the item can be longer than the specified text.

To find an item but not select it, use the FindItem function.

MultiSelect ListBoxes
SelectItem has no effect on a ListBox or PictureListBox whose MultiSelect property is true. Instead, use SetState to select items without affecting the selected state of other items in the list.

Clearing the edit box of a drop-down list
To clear the edit box of a DropDownListBox or DropDownPictureListBox that the user cannot edit, use Syntax 2 of SelectItem.

Examples

If item 5 in lb_Actions is Delete Files, this example starts searching after item 2, finds and highlights Delete Files, and sets *li_Index* to 5:

```
integer li_Index
li_Index = lb_Actions.SelectItem("Delete Files", 2)
```

If item 4 in lb_Actions is Select Objects, this example starts searching after item 2, finds and highlights Select Objects, and sets *li_Index* to 4:

```
integer li_Index
li_Index = lb_Actions.SelectItem("Sel", 2)
```

See also

AddItem
DeleteItem
FindItem
InsertItem
SetState

Syntax 2 When you know the item number

Description

Finds and highlights an item in a ListBox when you can specify the index number of the item. You can also clear the selection by specifying zero as the index number.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

ListBox, DropDownListBox, PictureListBox, and DropDownPictureListBox controls

Syntax

listboxname.**SelectItem** (*itemnumber*)

Argument	Description
listboxname	The name of the ListBox control in which you want to select an item
itemnumber	An integer whose value is the location (index) of the item in the ListBox or the ListBox portion of the drop-down list. Specify 0 for *itemnumber* to clear the selected item. For a ListBox or PictureListBox, 0 removes highlighting from the selected item. For a DropDownListBox or DropDownPictureListBox, 0 clears the text box.

Return value

Integer. Returns the index number of the selected item. SelectItem returns 0 if *itemnumber* is not valid or if you specified 0 in order to clear the selected item. It returns -1 if an error occurs. If any argument's value is null, SelectItem returns null.

Usage

To find an item but not select it, use the FindItem function.

MultiSelect ListBoxes
SelectItem has no effect on a ListBox or PictureListBox whose MultiSelect property is true. Instead, use SetState to select items without affecting the selected state of other items in the list.

Clearing the text box of a drop-down list
To clear the text box of a DropDownListBox or DropDownPictureListBox that the user cannot edit, set *itemnumber* to 0. Setting the control's text to the empty string does not work if the control's AllowEdit property is false.

Examples

This example highlights item number 5:

```
integer li_Index
li_Index = lb_Actions.SelectItem(5)
```

This example clears the selection from the text box of the DropDownListBox ddlb_choices and sets *li_Index* to 0:

```
integer li_Index
li_Index = ddlb_choices.SelectItem(0)
```

See also

AddItem
DeleteItem
FindItem

InsertItem
SetState

Syntax 3 For TreeView controls

Description
Selects a specified item.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to
TreeView controls

Syntax
treeviewname.**SelectItem** (*itemhandle*)

Argument	Description
treeviewname	The name of the TreeView control in which you want to select an item
itemhandle	The handle of the specified item

Return value
Integer. Returns 1 if it succeeds and -1 if an error occurs.

Usage
Use the FindItem function to get handles for items at specific positions in the TreeView control.

Examples
This example selects the parent of the current TreeView item:

```
long ll_tvi, ll_tvparent
int li_tvret
ll_tvi = tv_list.FindItem(CurrentTreeItem! , 0)
ll_tvparent = tv_list.FindItem(ParentTreeItem! , &
        ll_tvi)
li_tvret = tv_list.SelectItem(ll_tvparent)
```

See also
FindItem

SelectObject

Description
Selects or clears the object in an OLE control but does not activate the server application. The server's menus are added to the PowerBuilder application's menus.

PocketBuilder	✗
PowerBuilder	✓

Applies to
OLE controls

Syntax
olecontrol.**SelectObject** (*selectstate*)

Return value
Integer. Returns 0 if it succeeds and a negative number if an error occurs.

SelectTab

Description
Selects the specified tab, displaying its tab page in the Tab control.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✓

Applies to
Tab controls

Syntax
tabcontrolname.**SelectTab** (*tabidentifier*)

Argument	Description
tabcontrolname	The name of the Tab control in which you want to select a tab
tabidentifier	The tab you want to select. You can specify: • The tab page index (an integer) • The name of the user object (datatype DragObject or UserObject) • A string holding the name of the user object

Return value
Integer. Returns 1 if it succeeds and -1 if an error occurs.

Usage
Equivalent syntax You can select a tab by setting the SelectedTab property to the tab's index number:

```
tab_1.SelectedTab = 3
```

Examples

These three examples select the third tab in tab_1. They could be in the script for a CommandButton on the window containing the Tab control tab_1:

```
tab_1.SelectTab(3)
```

```
tab_1.SelectTab(tab_1.uo_3)
```

```
string ls_tabpage
ls_tabpage = "uo_3"
tab_1.SelectTab(ls_tabpage)
```

This example opens an instance of the user object uo_fontsettings as a tab page and selects it:

```
userobject uo_tabpage
string ls_tabpage
ls_tabpage = "uo_fontsettings"
tab_1.OpenTab(uo_tabpage, ls_tabpage, 0)
tab_1.SelectTab(uo_tabpage)
```

See also

OpenTab

SelectText

Selects text in an editable control.

To select text in	Use
Any editable control, other than a RichTextEdit	Syntax 1
A RichTextEdit control or a DataWindow whose object has the RichTextEdit presentation style	Syntax 2

Syntax 1 For editable controls (except RichTextEdit)

Description

Selects text in an editable control. You specify where the selection begins and how many characters to select.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

DataWindow, EditMask, MultiLineEdit, SingleLineEdit, DropDownListBox, and DropDownPictureListBox controls

Syntax

editname.**SelectText** (*start*, *length*)

Argument	Description
editname	The name of the DataWindow, EditMask, MultiLineEdit, SingleLineEdit, DropDownListBox, or DropDownPictureListBox control in which you want to select text.
start	A long specifying the position at which you want to start the selection.
length	A long specifying the number of characters you want to select. If *length* is 0, no text is selected but PocketBuilder moves the insertion point to the location specified in *start*.

Return value

Long. Returns the number of characters selected. If an error occurs, SelectText returns -1. If any argument's value is null, SelectText returns null.

Usage

If the control does not have the focus when you call SelectText, then the text is not highlighted until the control has focus. To set focus on the control so that the selected text is highlighted, call the SetFocus function.

How much to select
When you want to select all the text of a line edit or select the contents from a specified position to the end of the edit, use the Len function to obtain the length of the control's text.

To select text in a DataWindow with the RichTextEdit presentation style, use Syntax 2.

Examples

This statement sets the insertion point at the end of the text in the SingleLineEdit sle_name:

```
sle_name.SelectText(Len(sle_name.Text), 0)
```

This statement selects the entire contents of the SingleLineEdit sle_name:

```
sle_name.SelectText(1, Len(sle_name.Text))
```

The rest of these examples assume the MultiLineEdit mle_EmpAddress contains `Boston Street`.

The following statement selects the string `ost` and returns 3:

```
mle_EmpAddress.SelectText(2, 3)
```

The next statement selects the string `oston Street` and returns 12:

```
mle_EmpAddress.SelectText(2, &
      Len(mle_EmpAddress.Text))
```

These statements select the string Bos, returns 3, and sets the focus to mle_EmpAddress so that Bos is highlighted:

```
mle_EmpAddress.SelectText(1, 3)
mle_EmpAddress.SetFocus()
```

See also

Len
Position
SelectedItem
SelectedText
SetFocus
TextLine

Syntax 2 For RichTextEdit controls and presentation styles

Description

Selects text beginning and ending at a line and character position in a RichTextEdit control.

PocketBuilder	✗
PowerBuilder	✓

Applies to

RichTextEdit and DataWindow controls

Syntax

rtename.**SelectText** (*fromline*, *fromchar*, *toline*, *tochar* { *band* })

Return value

Long. Returns the number of characters selected. If an error occurs it returns -1. If any argument's value is null, SelectText returns null.

SelectTextAll

Description

Selects all the contents of a RichTextEdit control including any special characters such as a carriage return (CR), line feel (LF), and end-of-file (EOF).

PocketBuilder	✗
PowerBuilder	✓

Applies to

RichTextEdit and DataWindow controls

Syntax

rtename.**SelectTextAll** ({ *band* })

Return value

Integer. Returns the number of characters selected. If an error occurs, SelectTextAll returns -1.

SelectTextLine

Description
Selects the line containing the insertion point in a RichTextEdit control.

PocketBuilder	✗
PowerBuilder	✓

Applies to
RichTextEdit and DataWindow controls

Syntax
rtename.**SelectTextLine** ( )

Return value
Integer. Returns the number of characters selected if it succeeds and -1 if an error occurs.

SelectTextWord

Description
Selects the word containing the insertion point in a RichTextEdit control.

PocketBuilder	✗
PowerBuilder	✓

Applies to
RichTextEdit and DataWindow controls

Syntax
rtename.**SelectTextWord** ( )

Return value
Integer. Returns the number of characters selected if it succeeds and -1 if a word cannot be selected or an error occurs.

Send

Sends messages to a window, appointment notices to recipients, or SMS messages to a specified address.

To send	Use
A message to a window	Syntax 1
A Pocket Outlook appointment to a recipient	Syntax 2
An SMS message	Syntax 3

Syntax 1 For sending messages to a window

Description

Sends a message to a window so that it is executed immediately.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

Send (*handle*, *message#*, *lowword*, *long*)

Argument	Description
handle	A long whose value is the system handle of a window (that you have created in PocketBuilder or another application) to which you want to send a message.
message#	An UnsignedInteger whose value is the system message number of the message you want to send.
lowword	A long whose value is the integer value of the message. If this argument is not used by the message, enter 0.
long	The long value of the message or a string.

Return value

Long. Returns the value returned by SendMessage in Windows if it succeeds and -1 if an error occurs. If any argument's value is null, Send returns null.

Usage

PocketBuilder's Send function sends the message identified by *message#* and optionally, *lowword* and *long*, to the window identified by *handle* to the Windows function SendMessage. The message is sent directly to the object, bypassing the object's message queue. Send waits until the message is processed and obtains the value returned by SendMessage.

Messages in Windows
Use the Handle function to get the Windows handle of a PocketBuilder object.

You specify Windows messages by number. They are documented in the file *WINDOWS.H* that is part of the Microsoft Windows Software Development Kit (SDK) and other Windows development tools.

Posting a message
Messages sent with Send are executed immediately. To post a message to the end of an object's message queue, use the Post function.

Examples

This statement scrolls the window w_emp up one page:

```
Send(Handle(w_emp), 277, 2, 0)
```

Both of the following statements click the CommandButton cb_OK:

```
Send(Handle(Parent), 273, 0, Handle(cb_OK))

cb_OK.TriggerEvent(Clicked!)
```

You can send messages to maximize or minimize a DataWindow, and return it to normal. To use these messages, enable the TitleBar, Minimize, and Maximize properties of your DataWindow control. Also, you should give your DataWindow control an icon for its minimized state.

This statement minimizes the DataWindow:

```
Send(Handle(dw_whatever), 274, 61472, 0)
```

This statement maximizes the DataWindow:

```
Send(Handle(dw_whatever), 274, 61488, 0)
```

This statement returns the DataWindow to its normal, defined size:

```
Send(Handle(dw_whatever), 274, 61728, 0)
```

You can send a Windows message to determine the last item clicked in a multiselect ListBox. The following script for the SelectionChanged event of a ListBox control gets the return value of the LB_GETCURSEL message which is the item number in the list (where the first item is 0, not 1). To get PocketBuilder's index for the list item, the example adds 1 to the return value from Send. In this example, idx is an integer instance variable for the window:

```
// Send the Windows message for LB_GETCURSEL
// to the list box
idx = Send(Handle(This), 1033, 0, 0)
idx = idx + 1
```

See also

Handle
Post

Syntax 2 For POOMAppointment objects

Description

Sends the appointment (meeting request) to all recipients.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✗

Applies to

POOMAppointment objects

Syntax

Integer *objectname*.send ( )

Argument	Description
objectname	The name of the POOMAppointment or POOMTask object

Return value

Integer. Returns 1 for success and one of the following negative values if an error occurs:

-1	Unspecified error
-2	Cannot connect to the repository or a required internal subobject failed to connect to the repository
-3	Cannot log in to the repository
-4	Incorrect input argument
-5	Action cannot be performed
-6	The object identifier (OID) is not in the repository
-7	Feature is not implemented yet
-8	No matching entries found for the criteria

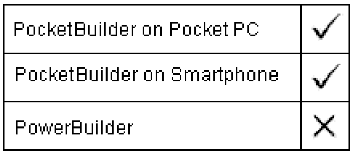
See also

AddToInfraredQueue

Syntax 3 For SMSSession objects

Description

Send an SMS message.



PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✗

Applies to

SMSSession objects

Syntax

objectname.Send (*smsmsg*, *destaddr*)

Argument	Description
objectname	The name of the SMSSession object
smsmsg	An SMSMessage structure returned by reference that contains information about the message
destaddr	An SMSAddress structure that contains the address to which the message should be sent

Return value

Integer. Returns 1 for success and a negative value if an error occurs.

Usage

The Send function sends an SMSMessage structure to an address specified in an SMSAddress structure.

Examples

The following example sets the text of the *g_smsMsg* SMSMessage structure from a multiline edit box, sets the address from a single-line edit, and sends the message to an international phone number:

```
// Global variables:
// SMSSession g_smsSess
// SMSMessage g_smsMsg
// SMSAddress g_smsmAddr

g_smsMsg.Text = mle_msg.text
g_smsAddr.AddressType = SMSAT_INTERNATIONAL!
g_smsAddr.Address = sle_addr.text
g_smsSess.Send(g_smsMsg, g_smsAddr)
```

See also

Open
GetMessageStatus

SendToInfrared

Description

Sends the entire infrared queue.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✗

Applies to

POOM objects

Syntax

Integer *objectname*.SendToInfrared ( )

Argument	Description
objectname	The name of the POOM object

Return value

Integer. Returns 1 for success and one of the following negative values if an error occurs:

-1	Unspecified error
-2	Cannot connect to the repository or a required internal subobject failed to connect to the repository

-3	Cannot log in to the repository
-4	Incorrect input argument
-5	Action cannot be performed
-6	The object identifier (OID) is not in the repository
-7	Feature is not implemented yet
-8	No matching entries found for the criteria

Usage

A user must be logged in to a POOM object to send an infrared queue. Calling SendToInfrared turns on the infrared beam and drains the queue.

Examples

The following example sends an infrared queue:

```
li_rtn = g_poom.SendToInfrared()
```

See also

AddToInfraredQueue
ReceiveFromInfrared

SeriesCount

Description

Counts the number of series in a graph.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Graph controls in windows and user objects, and graphs in DataWindow controls

Syntax

controlname.**SeriesCount** ({ *graphcontrol* })

Argument	Description
controlname	The name of the graph for which you want the number of series, or the name of the DataWindow control containing the graph
graphcontrol (DataWindow control only) (optional)	A string whose value is the name of the graph in the DataWindow control for which you want the number of series

Return value

Integer. Returns the number of series in the graph if it succeeds and -1 if an error occurs. If any argument's value is null, SeriesCount returns null.

Examples

These statements store in the variable *li_series_count* the number of series in the graph gr_product_data:

```
integer li_series_count
li_series_count = gr_product_data.SeriesCount()
```

These statements store in the variable *li_series_count* the number of series in the graph gr_computers in the DataWindow control dw_equipment:

```
integer li_series_count
li_series_count = &
      dw_equipment.SeriesCount("gr_computers")
```

See also

CategoryCount
DataCount

SeriesName

Description

Obtains the series name associated with the specified series number.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Graph controls in windows and user objects, and graphs in DataWindow controls

Syntax

controlname.**SeriesName** ({ *graphcontrol*, } *seriesnumber*)

Argument	Description
controlname	The name of the graph in which you want the name of a series, or the name of the DataWindow containing the graph
graphcontrol (DataWindow control only) (optional)	A string whose value is the name of the graph in the DataWindow control for which you want the name of a series
seriesnumber	The number of the series for which you want to obtain the name

Return value

String. Returns the name assigned to the series. If an error occurs, it returns the empty string (""). If any argument's value is null, SeriesName returns null.

Usage

Series are numbered consecutively, from 1 to the value returned by SeriesCount. When you delete a series, the series are renumbered to keep the numbering consecutive. You can use SeriesName to find out the name of the series associated with a series number.

Examples

These statements store in the variable *ls_SeriesName* the name of series 5 in the graph gr_product_data:

```
string ls_SeriesName
ls_SeriesName = gr_product_data.SeriesName(5)
```

These statements store in the variable *ls_SeriesName* the name of series 5 in the graph gr_computers in the DataWindow control dw_equipment:

```
string ls_SeriesName
ls_SeriesName = &
      dw_equipment.SeriesName("gr_computers", 5)
```

See also

CategoryName
DeleteSeries
FindSeries

SetAbort

Declares that a transaction on a transaction server should be rolled back.

To roll back a transaction	Use
For OLETxnObject objects	Syntax 1
For TransactionServer objects	Syntax 2

Syntax 1 For OLETxnObject objects

Description
Declares that the current transaction should be rolled back.

PocketBuilder	✗
PowerBuilder	✓

Applies to
OLETxnObject objects

Syntax
oletxnobject.**SetAbort** ( )

Return value
Integer. Returns 1 if it succeeds and -1 if an error occurs.

Syntax 2 For TransactionServer objects

Description
Declares that a component cannot complete its work for the current transaction and that the transaction should be rolled back. The component instance are deactivated when the method returns.

PocketBuilder	✗
PowerBuilder	✓

Applies to
TransactionServer objects

Syntax
transactionserver.**SetAbort** ( )

Return value
Integer. Returns 1 if it succeeds and -1 if an error occurs.

SetAlignment

Description	Sets the alignment of the selected paragraphs in a RichTextEdit control.

PocketBuilder	✗
PowerBuilder	✓

Applies to	RichTextEdit controls
Syntax	*rtename*.**SetAlignment** (*align*)
Return value	Integer. Returns 1 if it succeeds and -1 if an error occurs.

SetArgElement

Description	Sets the value in the specified argument element.

PocketBuilder	✗
PowerBuilder	✓

Applies to	Window ActiveX controls
Syntax	*activexcontrol*.**SetArgElement** (*index*, *argument*)
Return value	Integer. Returns 1 if the function succeeds and -1 if an error occurs.

SetAutomationLocale

Description	Sets the language to be used in automation programming for an OLE object. Call SetAutomationLocale if you have programmed automation commands in a language other than the user's locale.

PocketBuilder	✗
PowerBuilder	✓

Applies to	OLE objects
Syntax	*olename*.**SetAutomationLocale** (*language*, *sortorder*)
Return value	Integer. Returns 0 if it succeeds and -1 if an error occurs.

SetAutomationPointer

Description Sets the automation pointer of an OLEObject object to the value of the automation pointer of another object.

PocketBuilder	✗
PowerBuilder	✓

Applies to OLEObject

Syntax *oleobject*.**SetAutomationPointer** (*object*)

Return value Integer. Returns 0 if it succeeds and -1 if the object does not contain a valid OLE automation pointer.

SetAutomationTimeout

Description Sets the number of milliseconds that a PowerBuilder client waits before canceling an OLE procedure call to the server.

PocketBuilder	✗
PowerBuilder	✓

Applies to OLEObject objects

Syntax *oleobject*.**SetAutomationTimeout** (*interval*)

Return value Integer. Returns 0 if it succeeds and -1 if it fails.

SetCaptureImageAttributes

Description Sets image attributes such as picture size and zoom value for capturing a picture.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✗

Applies to Camera objects

Syntax

objectname.SetCaptureImageAttributes (*attrValue*)

Argument	Description
objectname	The name of the camera object for which you want to set capture attributes
attrValue	A CameraImageAttributes structure that contains the attributes to be set for the device

Return value

Integer. Returns 1 for success, and one of the following negative values if an error occurs:

-1	Unspecified error
-2	Supporting DLL not loaded
-3	Other initialization error
-5	Inconsistency in this object instance
-6	Call to the driver or device failed
-7	Unsupported option
-8	Value for option is out of range

Usage

You can set different attributes for previewing and capturing images. Typical capture values are 640 and 480 pixels for width and height and 2 for zoom.

Examples

This example gets the attributes that are available for a device in an array of CameraImageAttributes structures and displays them to the user so that the user can select the set of attributes to be used for preview and capture:

```
CameraImageAttributes AllowedConfigs[]
g_myCam.GetAllowedImageAttributes(AllowedConfigs)

// Display choices to user and let user select
// a preview and capture configuration
...
// User chose 1 for preview, 3 for capture
g_myCam.SetPreviewImageAttributes(AllowedConfigs[1])
g_myCam.SetCaptureImageAttributes(AllowedConfigs[3])
```

See also

CaptureImage
GetOption
IsReadyToCapture
Open
SetOption
SetPreviewImageAttributes

SetColumn

Description

Sets column information for a DataWindow, child DataWindow, or ListView control.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

For syntax for a DataWindow or child DataWindow, see the SetColumn method for DataWindows in the *DataWindow Reference* or the online Help.

Applies to

ListView controls

Syntax

listviewname.**SetColumn** (*index, label, alignment, width*)

Argument	Description
listviewname	The name of the ListView control for which you want to set column properties.
index	The number of the column for which you want to set column properties.
label	The label of the column for which you want to set column properties.
alignment	A value of the Alignment enumerated datatype specifying how to align the column. Values are: • Left! — Align the column at the left margin • Right! — Align the column at the right margin • Center! — Center the column between the left and right margins • Justify! — Not valid for the SetColumn function
width	The width of the column for which you want to set column properties.

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs.

Usage

SetColumn is used only in report views.

Examples

This example sets the second column of a ListView:

```
lv_list.SetColumn(2 , "Order" , Center! , 800)
```

See also

AddColumn
AddItem
SetItem

SetComplete

Declares that a transaction on a transaction server should be committed.

To commit a transaction	Use
For OLETxnObject objects	Syntax 1
For TransactionServer objects	Syntax 2

Syntax 1 For OLETxnObject objects

Description

Declares that the current transaction should be committed.

PocketBuilder	✗
PowerBuilder	✓

Applies to

OLETxnObject objects

Syntax

oletxnobject.**SetComplete** ( )

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs.

Syntax 2 For TransactionServer objects

Description

Declares that the transaction in which a component is participating should be committed and the component instance should be deactivated.

PocketBuilder	✗
PowerBuilder	✓

Applies to

TransactionServer objects

Syntax

transactionserver.**SetComplete** ( )

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs.

SetData

Description
Sets data in the OLE server associated with an OLE control using Uniform Data Transfer.

PocketBuilder	✗
PowerBuilder	✓

Applies to
OLE controls and OLE custom controls

Syntax
olename.**SetData** (*clipboardformat*, *data*)

Return value
Integer. Returns 0 if it succeeds and -1 if an error occurs.

SetDataAsInk

Description
Sets the data in the control in Pocket Word Ink (PWI) format. This format is compatible with Pocket Word.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✗

Applies to
Signature controls

Syntax
Integer *controlname*.SetDataAsInk (*data*)

Argument	Description
controlname	The name of the control for which you want to set the data
data	The blob containing the data in PWI format

Return value
Integer. Returns 1 for success and a negative integer for failure.

Usage
The SetDataAsInk function can set both typed and freehand drawing or writing into a Signature control.

Examples
The following example reads data in PWI format from a file into a blob, then sets the data into a Signature control:

```
blob lblb_ink
integer li_file, li_rtn

li_file = FileOpen("\My Documents\testpwi.pwi", &
   StreamMode!, Read!)
FileRead( li_file, lblb_ink )
FileClose( li_file )

li_rtn = sig_1.SetDataAsInk(lblb_ink)
sle_1.text = string(li_rtn)
```

See also

GetDataAsRTF
SetDataAsInk
SetDataAsText

SetDataAsRTF

Description

Sets the contents of a control from the data in a string. The text formatting in the string is maintained in the control.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✗

Applies to

Signature controls

Syntax

Integer *controlname*.SetDataAsInk (*data*)

Argument	Description
controlname	The name of the control for which you want to set the data
data	The blob or Unicode string containing the data

Return value

Integer. Returns 1 for success and a negative integer for failure.

Usage

The SetDataAsRTF function can set only text data into a Signature control.

Examples

The following example reads data in RTF format from a file into a blob, then sets the data into a Signature control:

```
   blob lblb_rtf
   integer li_file, li_rtn

   li_file = FileOpen("\My Documents\testb.rtf", &
      StreamMode!, Read!)
   FileRead( li_file, lblb_rtf )
   FileClose( li_file )

   li_rtn = sig_1.SetDataAsRTF(lblb_rtf)
```

The following example reads data in RTF format from a file into a Unicode string, then sets the data into a Signature control:

```
string ls_rtf
integer li_file, li_rtn

li_file = FileOpen("\My Documents\tests.rtf",  &
   StreamMode!, Read!)
FileRead( li_file, ls_rtf )
FileClose( li_file )

li_rtn = sig_1.SetDataAsRTF(ls_rtf)
```

See also

GetDataAsRTF
SetDataAsInk
SetDataAsText

SetDataAsText

Description

Formats data in a control as plain text.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✗

Applies to

Signature controls

Syntax

Integer *controlname*.SetDataAsText (*data*)

Argument	Description
controlname	The name of the control for which you want to set the data
data	The Unicode string containing the text data

Return value

Integer. Returns 1 for success and a negative integer for failure.

Usage

The SetDataAsText function can set only text data into a Signature control.

Examples

The following example reads data in text format from a file into a Unicode string, then sets the data into a Signature control:

```
string ls_txt
integer li_file, li_rtn

li_file = FileOpen("\My Documents\tests.txt",  &
   StreamMode!, Read!)
```

```
FileRead( li_file, ls_txt )
FileClose( li_file )

li_rtn = sig_1.SetDataAsText(ls_txt)
```

See also

GetDataAsText
SetDataAsInk
SetDataAsRTF

SetDataDDE

Description

Sends data to a DDE client application when PowerBuilder is acting as a DDE server. You would usually call SetDataDDE in the script for the RemoteRequest event, which is triggered by a DDE request for data from the client application.

PocketBuilder	✗
PowerBuilder	✓

Syntax

SetDataDDE (*string* {, *applname*, *topic*, *item* })

Return value

Integer. Returns 1 if it succeeds. If an error occurs, SetDataDDE returns a negative integer.

SetDataPieExplode

Description

Explodes a pie slice in a pie graph. The exploded slice is moved away from the center of the pie, which draws attention to the data. You can explode any number of slices of the pie.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Graph controls in windows and user objects, and graphs in DataWindow controls

Syntax

controlname.**SetDataPieExplode** ({ *graphcontrol*, } *seriesnumber*, *datapoint*, *percentage*)

Argument	Description
controlname	The name of the graph in which you want to explode a pie slice, or the name of the DataWindow containing the graph.
graphcontrol (DataWindow control only) (optional)	A string whose value is the name of the graph in the DataWindow control in which you want to explode a pie slice.
seriesnumber	The number that identifies the series.
datapoint	The number of the data point (that is, the pie slice) to be exploded.
percentage	A number between 0 and 100 which is the percentage of the radius that the pie slice is moved away from the center. When *percentage* is 100, the tip of the slice is even with the circumference of the pie's circle.

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is null, SetDataPieExplode returns null.

Usage

If the graph is not a pie graph, the function has no effect.

Examples

This example explodes the pie slice under the pointer to 50% when the user double-clicks within the graph. The code checks the property GraphType to make sure the graph is a pie graph. It then finds out whether the user clicked on a pie slice by checking the series and data point values set by ObjectAtPointer. The script is for the DoubleClicked event of a graph object:

```
integer series, datapoint
grObjectType clickedtype
integer percentage

percentage = 50
IF (This.GraphType <> PieGraph! AND &
       This.GraphType <> Pie3D!) THEN RETURN
clickedtype = This.ObjectAtPointer( &
       series, datapoint)
IF (series > 0 and datapoint > 0) THEN
       This.SetDataPieExplode(series, datapoint, &
          percentage)
END IF
```

See also

GetDataPieExplode

SetDataStyle

Specifies the appearance of a data point in a graph. The data point's series has appearance settings that you can override with SetDataStyle.

To	Use
Set the data point's colors	Syntax 1
Set the line style and width for the data point	Syntax 2
Set the fill pattern or symbol for the data point	Syntax 3

Syntax 1 For setting a data point's colors

Description

Specifies the colors of a data point in a graph.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Graph controls in windows and user objects, and graphs in DataWindow controls

Syntax

controlname.**SetDataStyle** ({ *graphcontrol*, } *seriesnumber*,
datapointnumber, *colortype*, *color*)

Argument	Description
controlname	The name of the graph in which you want to set the color of a data point, or the DataWindow containing the graph.
graphcontrol (DataWindow control only) (optional)	A string whose value is the name of the graph in the DataWindow control in which you want to set the color of a data point.
seriesnumber	The number of the series in which you want to set the color of a data point.
datapointnumber	The number of the data point for which you want to set the color.

Argument	**Description**
colortype	A value of the grColorType enumerated datatype specifying the aspect of the data point for which you want to set the color. Values are: • Foreground! — Text color • Background! — Background color • LineColor! — Line color • Shade! — Shade (for graphics that are three-dimensional or have solid objects)
color	A long whose value is the new color for *colortype*.

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is null, SetDataStyle returns null.

Usage

To change the appearance of a series, use SetSeriesStyle. The settings you make for the series are the defaults for all data points in the series.

To reset the color of individual points back to the series color, call ResetDataColors.

For a graph in a DataWindow, you can specify the appearance of a data point in the graph before PocketBuilder draws the graph. To do so, define a user event for pbm_dwngraphcreate and call SetDataStyle in the script for that event. The event pbm_dwngraphcreate is triggered just before a graph is created in a DataWindow object.

Examples

This example checks the background color for data point 6 in the series named Salary in the graph gr_emp_data. If it is red, SetDataStyle sets it to black:

```
long color_nbr
integer SeriesNbr
// Get the number of the series
SeriesNbr = gr_emp_data.FindSeries("Salary")
// Get the background color
gr_emp_data.GetDataStyle(SeriesNbr, 6, &
      Background!, color_nbr)
// If color is red, change it to black
IF color_nbr = 255 THEN &
      gr_emp_data.SetDataStyle(SeriesNbr, 6, &
         Background!, 0)
```

These statements set the text (foreground) color to black for data point 6 in the series named Salary in the graph gr_depts in the DataWindow control dw_employees:

```
integer SeriesNbr
```

```
// Get the number of the series
SeriesNbr = &
      dw_employees.FindSeries("gr_depts" , "Salary")
// Set the background color
dw_employees.SetDataStyle("gr_depts" , SeriesNbr, &
      6, Background!, 0)
```

See also

GetDataStyle
GetSeriesStyle
ResetDataColors
SeriesName
SetSeriesStyle

Syntax 2 For the line associated with a data point

Description

Specifies the style and width of a data point's line in a graph.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Graph controls in windows and user objects, and graphs in DataWindow controls

Syntax

controlname.**SetDataStyle** ({ *graphcontrol*, } *seriesnumber*, *datapointnumber*, *linestyle*, *linewidth*)

Argument	Description
controlname	The name of the graph in which you want to set the line style and width of a data point, or the name of the DataWindow containing the graph.
graphcontrol (DataWindow control only) (optional)	A string whose value is the name of the graph in the DataWindow control in which you want to set the line style and width.
seriesnumber	The number of the series in which you want to set the line style and width of a data point.
datapointnumber	The number of the data point for which you want to set the line style and width.

Argument	Description
linestyle	A value of the LineStyle enumerated datatype. Values are: Continuous! Dash! DashDot! DashDotDot! Dot! Transparent!
linewidth	An integer whose value is the width of the line in pixels.

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is null, SetDataStyle returns null.

Usage

To change the appearance of a series, use SetSeriesStyle. The settings you make for the series are the defaults for all data points in the series.

For a graph in a DataWindow, you can specify the appearance of a data point in the graph before PocketBuilder draws the graph. To do so, define a user event for pbm_dwngraphcreate and call SetDataStyle in the script for that event. The event pbm_dwngraphcreate is triggered just before a graph is created in a DataWindow object.

Examples

This example checks the line style used for data point 10 in the series named Costs in the graph gr_computers in the DataWindow control dw_equipment. If it is dash-dot, the SetDataStyle sets it to continuous. The line width stays the same:

```
integer SeriesNbr, line_width
LineStyle line_style

// Get the number of the series
SeriesNbr = dw_equipment.FindSeries( &
      "gr_computers", "Costs")

// Get the current line style
dw_equipment.GetDataStyle("gr_computers", &
      SeriesNbr, 10, line_style, line_width)

// If the pattern is dash-dot, change to continuous
IF line_style = DashDot! THEN &
      dw_equipment.SetDataStyle("gr_computers", &
         SeriesNbr, 10, Continuous!, line_width)
```

See also

GetDataStyle
GetSeriesStyle
SeriesName
SetSeriesStyle

Syntax 3 For the fill pattern and symbol of a data point

Description

Specifies the fill pattern and symbol for a data point in a graph.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Graph controls in windows and user objects, and graphs in DataWindow controls

Syntax

controlname.**SetDataStyle** ({ *graphcontrol*, } *seriesnumber*,
    datapointnumber, *enumvalue*)

Argument	Description
controlname	The name of the graph in which you want to set the appearance of a data point, or the name of the DataWindow containing the graph.
graphcontrol (DataWindow control only) (optional)	A string whose value is the name of the graph in the DataWindow control in which you want to set the appearance.
seriesnumber	The number of the series in which you want to set the appearance of a data point.
datapointnumber	The number of the data point for which you want to set the appearance.

Argument	Description
enumvalue	An enumerated datatype specifying the appearance setting for the data point. You can specify a FillPattern or grSymbolType value. To change the fill pattern, use a FillPattern value: Bdiagonal! — Lines from lower left to upper right Diamond! Fdiagonal! — Lines from upper left to lower right Horizontal! Solid! Square! Vertical! To change the symbol type, use a grSymbolType value: NoSymbol! SymbolHollowBox! SymbolX! SymbolStar! SymbolHollowUpArrow! SymbolHollowCircle! SymbolHollowDiamond! SymbolSolidDownArrow! SymbolSolidUpArrow! SymbolSolidCircle! SymbolSolidDiamond! SymbolPlus! SymbolHollowDownArrow! SymbolSolidBox!

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is null, SetDataStyle returns null.

Usage

To change the appearance of a series, use SetSeriesStyle. The settings you make for the series are the defaults for all data points in the series.

For a graph in a DataWindow, you can specify the appearance of a data point in the graph before PocketBuilder draws the graph. To do so, define a user event for pbm_dwngraphcreate and call SetDataStyle in the script for that event. The event pbm_dwngraphcreate is triggered just before a graph is created in a DataWindow object.

Examples

This example checks the fill pattern used for data point 10 in the series named Costs in the graph gr_product_data. If it is diamond, then SetDataStyle changes it to solid:

```
integer SeriesNbr
FillPattern data_pattern
```

```
// Get the number of the series
SeriesNbr = gr_product_data.FindSeries("Costs")

// Get the current fill pattern
gr_product_data.GetDataStyle(SeriesNbr, 10, &
      data_pattern)

// If the pattern is diamond, change it to solid
IF data_pattern = Diamond! THEN &
      gr_product_data.SetDataStyle(SeriesNbr, &
         10, Solid!)
```

See also

GetDataStyle
GetSeriesStyle
SeriesName
SetSeriesStyle

SetDisplayZoom

Description

Sets the zoom factor of controls as a percent of their size at design time.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✗

Syntax

SetDisplayZoom (*izoom*, *rebuild*)

Argument	Description
izoom	Integer for the zoom factor that you want to set for all application controls.
rebuild	Boolean that determines whether the zoom factor applies to new controls only or to all controls in the application. Values are: • **true** All controls are resized • **false** Only controls in new windows are resized

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs.

Usage

The current zoom factor that you set in a SetDisplayZoom call applies to all controls (when *rebuild* = true) or all newly instantiated controls (when *rebuild* = false) in an application. However, when *rebuild* = true, the sizes of the bitmaps for radio buttons, check boxes, and the edit boxes of drop-down lists are not changed by a SetDisplayZoom call unless they are used as display formats for columns in a DataWindow. When the same controls are placed on application windows, the sizes of these controls' bitmaps are fixed by the Windows CE operating system and can be modified only by a SetDisplayZoom call before they are loaded.

Setting the zoom factor for windows with DataWindow controls
The zoom value should be set before any dynamic changes are made to the DataWindow content since changing the display zoom value resets the DataWindow content.

The zoom factor is a percent of the size of the controls at design time. SetDisplayZoom works best for devices that have a VGA screen, such as the ASUS MyPal A730. The permissible zoom factor range is 10 to 500 percent. If you set a zoom factor outside of this range, PocketBuilder automatically resets the zoom factor to 100.

Although horizontal and vertical scroll bars are resized based on the zoom factor that you set in a SetDisplayZoom call, a threshold exists beyond which these controls cannot be painted. The threshold depends on the device resolution. For example, a scroll bar is not visible on a Dell Axim device with a 240 x 320 screen resolution if its height is less than 28 pixels.

Drawing objects, such as lines and ovals, are automatically repainted with the current zoom factor when an action causes the application window to be refreshed. This occurs even if you called SetDisplayZoom with the *rebuild* argument set to false.

See also

GetDisplayZoom

SetDropHighlight

Description

Highlights the specified item as the drop target.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

TreeView controls

Syntax

treeviewname.**SetDropHighlight** (*itemhandle*)

Argument	Description
treeviewname	The TreeView control in which you want to highlight an item as the target of a drag-and-drop operation
itemhandle	The handle of the item you want to highlight as the target in a drag-and-drop operation

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs.

Usage

Use in a drag operation to specify a drop target.

Examples

This example uses the TreeView Clicked event to set the current TreeView item as the drop target:

```
handle = tv_list.FindItem(CurrentTreeItem!,0)
tv_list.SetDropHighlight(handle)
```

See also

FindItem
SetItem

SetDynamicParm

Description

Specifies a value for an input parameter in the DynamicDescriptionArea that is used in an SQL OPEN or EXECUTE statement.

PocketBuilder	✗
PowerBuilder	✓

Syntax

DynamicDescriptionArea.**SetDynamicParm** (*index*, *value*)

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is null, SetDynamicParm returns null.

SetEndOfFile

Description

Sets the current position in a file as the last position in the file.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✗

Applies to

FileDirect objects

Syntax

instancename.SetEndOfFile ( )

Argument	Description
instancename	Instance name of the FileDirect object

Return value

Integer. Returns 1 for success and a negative number for an error.

Usage

Use the SetEndOfFile function to reset the current file position as the last position in the file.

Examples

The following script moves the file pointer 100 bytes from the file end before the Read function is called:

```
li_ret = nvo_FileDirect.setendoffile ( )
```

See also

Open
Seek

SetFirstVisible

Description

Sets the specified item as the first visible item in a TreeView control.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

TreeView controls

Syntax

treeviewname.**SetFirstVisible** (*itemhandle*)

Argument	Description
treeviewname	The TreeView control in which you want to identify an item as the first visible item

Argument	Description
itemhandle	The handle of the item you are identifying as the first visible item in the TreeView control

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs.

Usage

Use to give focus to the TreeView item specified by the *itemhandle* and scroll it to the top of the TreeView control (or as close to the top as the item list allows; if the item is the last item in a TreeView control, for example, it cannot scroll to the top of the control).

Examples

This example sets the current TreeView item as the first item visible in a TreeView control:

```
long ll_tvi
int li_tvret

ll_tvi = tv_list.FindItem(CurrentTreeItem! , 0)

li_tvret = tv_list.SetFirstVisible(ll_tvi)
IF li_tvret = -1 THEN
      MessageBox("Warning!" , "Didn't Work")
END IF
```

See also

FindItem
SetItem

SetFocus

Description

Sets the focus on the specified object or control.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Any object

Syntax

objectname.**SetFocus** ( )

Argument	Description
objectname	The name of the object or control in which you want to set the focus

Return value
: Integer. Returns 1 if it succeeds and -1 if an error occurs. If *objectname* is null, SetFocus returns null.

Usage
: If *objectname* is a ListBox, SetFocus displays the focus rectangle around the first item. If *objectname* is a DropDownListBox, SetFocus highlights the edit box. To select an item in a ListBox or DropDownListBox, use SelectItem.

Drawing objects cannot have focus. Therefore, you cannot use SetFocus to set focus to in a Line, Oval, Rectangle, or RoundRectangle.

Examples
: This statement in the script for the Open event in a window moves the focus to the first item in lb_Actions:

```
lb_Actions.SetFocus()
```

See also
: SetItem
SetState
SetTop

SetGlobalProperty

Description
: Sets the value of an SSL global property.

PocketBuilder	✗
PowerBuilder	✓

Applies to
: SSLServiceProvider object

Syntax
: *sslserviceprovider*.**SetGlobalProperty** (*property, value*)

Return value
: Long. Returns 0 for success and a negative number if an error occurs.

SetHold

Description
: Allows the user to put the current call on hold or retrieve a call that is on hold.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✗

Applies to
: PhoneCall objects

Syntax

objectname.SetHold (*holdstate*)

Argument	Description
objectname	The name of the PhoneCall object.
holdstate	A boolean that indicates whether the call is to be put on hold or retrieved from hold. Values are: • **true** Hold the current call • **false** Retrieve a call that is on hold

Return value

Integer. Returns a value that indicates the state of the phone call before SetHold is called, if it succeeds. Returns a negative value if an error occurs. Values are:

- **1** On hold
- **2** Not on hold

Examples

The following script for a Hold button puts a call on hold if it was not on hold and retrieves a call from hold if it was on hold:

```
// Global variable: Long g_phInit = 0
// set to 1 in pcall_1 constructor
// Global variable gb_holdstate
integer li_ret
if ( g_phInit > 0) then
   li_ret = pcall_1.SetHold( gb_holdstate)
   if (gb_holdstate = true) then
      gb_holdstate = false
   else
      gb_holdstate = true
   end if
else
   sle_1.text = "Call not initialized"
end if
```

See also

AcceptCall
AllowReceivingCalls
DropCall
MakeCall
SetMute
SetRingTone

SetItem

Sets the value of an item in a list.

For use with DataWindows and DataStores, see the SetItem method for DataWindows in the *DataWindow Reference* or the online Help.

To set the values of	Use
A ListView control item	Syntax 1
A ListView control item and column	Syntax 2
A TreeView control item	Syntax 3

Syntax 1 For ListView controls

Description

Sets data associated with a ListView item to the property values you specify in a ListViewItem variable.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

ListView controls

Syntax

listviewname.**SetItem** (*index*, { *column* }, *item*)

Argument	Description
listviewname	The ListView for which you are setting item properties
index	The index number of the item for which you are setting properties
column	The index number of the column of the item for which you want to set properties
item	The ListViewItem variable containing property values you want to assign to a ListView item

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs.

Usage

You can set properties for any ListView item with this syntax. If you do not specify a column, SetItem sets properties for the first column of an item. Only report views display multiple columns.

To add items to a ListView control, use the AddItem function. To add columns to a ListView control, use AddColumn. To set display values for the columns of a ListView item, use Syntax 2.

If you want to set column properties, such as alignment or width, use SetColumn. These column properties are independent of the ListViewItem objects.

To change pictures and other property values associated with a ListView item, use GetItem, change the property values, and use SetItem to apply the changes back to the ListView.

Examples

This example uses SetItem to change the state picture index for the selected lv_list ListView item:

```
listviewitem lvi_1

lv_list.GetItem(lv_list.SelectedIndex( ), lvi_1)
lvi_1.StatePictureIndex = 2
lv_list.SetItem(lv_list.SelectedIndex () , lvi_1)
```

See also

AddColumn
AddItem
GetItem
SetColumn

Syntax 2 For ListView controls

Description

Sets the value displayed for a particular column of a ListView item.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

ListView control

Syntax

listviewname.**SetItem** (*index, column, label*)

Argument	Description
listviewname	The ListView control for which you are setting a display value
index	The index number of the item for which you are setting a display value
column	The index number of the column for which you want to set a display value
label	The string value or variable which you are assigning to the specified column of the specified ListView item

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs.

Usage

You must include the column number as an argument, even if you are only assigning values to a single-column ListView control. To specify the properties for a ListView item, use Syntax 1.

Examples

This example assigns display values to three columns in a report view for three lv_list ListView items:

```
listviewitem l_lvi
integer li_count, li_index

FOR li_index = 1 to 3
      li_count=li_count+1
      lv_1ist.AddItem("Category " + String(li_index),
1)
NEXT

lv_list.AddColumn("Composition", Left! , 860)
lv_list.AddColumn(" Album", Left! , 610)
lv_list.AddColumn(" Artist", Left! , 710)

lv_list.SetItem(1 , 1 , "St. Thomas")
lv_list.SetItem(1 , 2 , "The Bridge")
lv_list.SetItem(1 , 3 , "Sonny Rollins")

lv_list.SetItem(2 , 1 , "So What")
lv_list.SetItem(2 , 2 , "Kind of Blue")
lv_list.SetItem(2 , 3 , "Miles Davis")

lv_list.SetItem(3 , 1 , "Goodbye, Porkpie Hat")
lv_list.SetItem(3 , 2 , "Mingus-Ah-Um")
lv_list.SetItem(3 , 3 , "Charles Mingus")
```

See also

GetItem

Syntax 3 For TreeView controls

Description

Sets the data associated with a specified item.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

TreeView controls

Syntax

treeviewname.**SetItem** (*itemhandle, item*)

Argument	Description
treeviewname	The name of the TreeView control in which you want to set the data for a specific item
itemhandle	The handle associated with the item you want to change
item	The TreeView item you want to change

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs.

Usage

Typically, you would call GetItem first, edit the data, and then call SetItem to reflect your changes in the TreeView control.

Examples

This example uses the ItemExpanding event to change the picture index and selected picture index of the current TreeView item:

```
treeviewitem l_tvi
long ll_tvi

ll_tvi = tv_list.FindItem(CurrentTreeItem! , 0)
tv_list.GetItem(ll_tvi , l_tvi)
l_tvi.PictureIndex = 5
l_tvi.SelectedPictureIndex = 5

tv_list.SetItem( ll_tvi, l_tvi )
```

See also

GetItem

SetItemPictureIndex

Description

Sets the picture for the item index of a toolbar item.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✗

Applies to

Toolbar controls

Syntax

Integer *controlname*.GetItemPictureIndex (*toolbarindex*, *pictureindex*)

Argument	Description
controlname	The name of the toolbar control
toolbarindex	Integer for the index of a toolbar item
pictureindex	Integer for the index of the picture you want to set for a toolbar item

Return value

Integer. Returns 1 for success and -1 if an error occurs.

Examples

The following example sets the picture index for the second item in the toolbar, assigning it the first picture in the toolbar picture name array:

```
Integer li_rtn
li_rtn = tlbr_mytoolbar.SetItemPictureIndex(2, 1)
```

See also

GetItemPictureIndex

SetItemState

Description

Sets the state of a toolbar item.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✗

Applies to

Toolbar controls

Syntax

Integer *controlname*.SetItemState (*toolbarindex*, *itemstate*)

Argument	Description
controlname	The name of the toolbar control
toolbarindex	Integer for the index of the toolbar item

Argument	Description
itemstate	Integer value to indicate the state of the toolbar item that you want to set. Values are: • **1** Sets a StyleCheck! or StyleCheckGroup! toolbar button in the depressed state • **2** Sets a StyleButton! toolbar button in a transitional depressed state • **4** Enables a toolbar item for selection • **32** Sets the next item in the toolbar on a separate line if it is not part of the same group Values are additive. For example, suppose you want to set a toolbar button with the checked state (1) and enable it (4), with the next set of buttons wrapped to a different line (32). You would enter 37 for the *itemstate* argument.

Return value

Integer. Returns 1 for success and -1 if an error occurs.

Examples

The following example sets the state for the second item in the toolbar:

```
Integer li_rtn
li_rtn = tlbr_mytoolbar.SetItemState(2, 33)
```

See also

GetItemState

SetLevelPictures

Description

Sets the picture indexes for all items at a particular level.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

TreeView controls

Syntax

treeviewname.**SetLevelPictures** (*level, pictureindex, selectedpictureindex, statepictureindex, overlaypictureindex*)

Argument	Description
treeviewname	The TreeView control in which you want to set the pictures for a given TreeView level
level	The TreeView level for which you are setting the picture indexes

Argument	Description
pictureindex	An index from the regular picture list specifying the picture to be displayed when the item is not selected
selectedpictureindex	An index from the regular picture list specifying the picture to be displayed when the item is selected
statepictureindex	An index from the state picture list specifying the picture to be displayed to the left of the regular picture
overlaypictureindex	An index from the overlay picture list specifying the picture to be displayed on top of the regular picture

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs.

Usage

To set pictures for individual items, call GetItem, set the picture properties, and call SetItem to copy the changes to the TreeView. You must specify a value for all four indexes. To display nothing, specify 0.

Examples

This example sets the pictures for TreeView level 3, then inserts two new TreeView items:

```
long ll_tvi, ll_child, ll_child2
int li_pict, li_level
treeviewitem l_tvi

li_level = 6
tv_list.SetLevelPictures( 3, li_level, li_level, &
      li_level, li_level)

ll_tvi = tv_list.FindItem(RootTreeItem! , 0)
ll_child = tv_list.InsertItemLast(ll_tvi, "Walton",2)
ll_child2 = tv_list.InsertItemLast(ll_child, &
      "Spitfire Suite", li_level)
tv_list.ExpandItem(ll_child)
tv_list.SetFirstVisible(ll_child)
```

See also

AddPicture

SetLibraryList

Description

Changes the files in the library search path of the application at runtime.

PocketBuilder	✗
PowerBuilder	✓

Obsolete syntax
You can still use the old syntax with the name of the application object before the function call: *applicationname*.SetLibraryList (*filelist*).

Syntax

SetLibraryList (*filelist*)

Return value

Integer. Returns 1 if it succeeds. If an error occurs, it returns:

-1 The application is being run from PowerBuilder, rather than from a standalone executable.

-2 A currently instantiated object is in a library that is not on the new list. If any argument's value is null, SetLibraryList returns null.

SetMask

Description

Sets the edit mask and edit mask datatype for an EditMask control.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

EditMask controls

Syntax

editmaskname.**SetMask** (*maskdatatype*, *mask*)

Argument	Description
editmaskname	The name of the EditMask for which you want to specify the edit mask.
maskdatatype	A MaskDataType enumerated datatype indicating the datatype of the mask. Values are: • DateMask! • DateTimeMask! • DecimalMask! • NumericMask! • StringMask! • TimeMask!
mask	A string whose value is the edit mask.

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is null, SetMask returns null.

Usage

In an edit mask, a fixed set of characters represent a type of character that the user can enter. In addition, punctuation controls the format of the entered value. Each mask datatype has its own set of valid characters.

For example, the following is a mask of type string for a telephone number. The EditMask control displays the punctuation (the parentheses and dash). The pound signs represent the digits that the user enters. The user cannot enter any characters other than digits.

```
(###) ###-####
```

For help in specifying a valid mask, see the Edit Mask Style dialog box for an EditMask control in the Window painter. A ListBox in the dialog box shows the meaning of the special mask characters for each datatype, as well as masks that have already been defined.

If you are specifying the mask for a number, the format must use U.S. notation. That is, comma represents the thousands delimiter and a period represents the decimal place. During execution, the locally correct symbols are displayed.

You cannot use color for edit masks as you can for display formats.

Examples

These statements set the mask for the EditMask password_mask to the mask in *pword_code*. The mask requires the user to enter a digit followed by four characters of any type:

```
string pword_code
pword_code = "#xxxx"
password_mask.SetMask(StringMask!, pword_code)
```

This statement sets the mask for the EditMask password_mask to a 5-digit numeric mask:

```
password_mask.SetMask(NumericMask!, "#####")
```

SetMessage

Description

Sets an error message for an object of type Throwable.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

throwableobject.**SetMessage** (*newMessage*)

Argument	Description
throwableobject	Object of type Throwable for which you want to set an error message.
newMessage	String containing the message you want to set. Must be surrounded by quotation marks.

Return value

None

Usage

Use to set a customized message on a user-defined exception object. Although it is possible to use SetMessage to modify the preset error messages for RuntimeError objects, this is not recommended.

Examples

This statement is an example of a message set on a user object of type Throwable:

```
MyException.SetMessage ("MyException thrown")
```

This example uses SetMessage in the try-catch block for a user-defined function that takes an input value from one text box and outputs the arccosine for that value into another text box:

```
uo_exception lu_error
Double ld_num
ld_num = Double (sle_1.text)

TRY
sle_2.text = string (acos (ld_num))
CATCH (runtimeerror er)
   lu_error = Create uo_exception
   lu_error.SetMessage("Value must be between -1" +&
      "and 1")
   Throw lu_error
END TRY
```

See also

GetMessage

SetMessageSink

Description

Specifies a visual object that will receive event notifications from user events with the pbm_command event ID.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✗

Applies to

NotificationBubble objects

Syntax

Integer *controlname*.SetMessageSink (*sinkWindow*)

Argument	**Description**
controlname	The name of the notification bubble that has a user event with the pbm_command event ID
sinkWindow	GraphicObject that you want to have receive event notifications

Return value

Integer. Returns 1 for success and -1 if an error occurs.

Usage

The NotificationBubble object contains HTML text that can include input controls with a command ID and number as a name attribute. For example, the HTML text could include the following command button element:

```
<input type=button name="cmd:10" value="OK">
```

If you create a user event on the NotificationBubble object with a pbm_command event ID, the event will be triggered when an application user taps the OK command button. Notification of the user action will be set to the visual object that you name in the *sinkWindow* argument.

Command IDs in the NotificationBubble's HTML text typically have values of 3 or greater. A value of cmd:1 sends a notification, but does not close the notification bubble. A value of cmd:2 closes the notification bubble but does not remove the notification from the notification tray, making it ideally suitable as the command ID for a Cancel button.

Examples

The following example sets the parent window of a notification bubble object to receive notifications from a user event:

```
li_rtn = nb_myBubble.SetMessageSink(parent)
```

See also

Icon
Update

SetMicroHelp

Description	Specifies the text to be displayed in the MicroHelp box in an MDI frame window.

PocketBuilder	✗
PowerBuilder	✓

Applies to	MDI frame windows
Syntax	*windowname*.**SetMicroHelp** (*string*)
Return value	Integer. Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is null, SetMicroHelp returns null.

SetMute

Description	Allows the user to mute or unmute the line.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✗

Applies to	PhoneCall objects
Syntax	*objectname*.SetMute (*mutestate*)

Argument	Description
objectname	The name of the PhoneCall object
mutestate	A boolean that indicates whether the line is to be muted or unmuted. Values are: • **true** Mute the line • **false** Unmute the line

Return value

Integer. Returns a value that indicates the state of the phone line before SetMute is called if it succeeds and a negative value if an error occurs. Values are:

- **1** Muted
- **2** Not muted

Examples

The following script for a Mute button mutes a line if it was muted and unmutes it if it was not muted:

```
// Global variable: Long g_phInit = 0
// set to 1 in pcall_1 constructor
// Global variable gb_mutestate
integer li_ret
if ( g_phInit > 0) then
   li_ret = pcall_1.SetMute( gb_mutestate)
   if (gb_mutestate = true) then
      gb_mutestate = false
   else
      gb_mutestate = true
   end if
else
   sle_1.text = "Call not initialized"
end if
```

See also

AcceptCall
AllowReceivingCalls
DropCall
MakeCall
SetHold
SetRingTone

SetNewMobiLinkPassword

Description

Reserved for future use. Sets a new password for the current MobiLink user.

Applies to

MLSynchronization, MLSync controls

Syntax

syncObject.SetNewMobiLinkPassword (*newPW*)

Argument	Description
syncObject	The name of the synchronization object that starts a connection to the synchronization server.
newPW	A string consisting of the new password that you want to set for MobiLink.

Return value

Integer. Returns 1 for succes and -1 for failure.

SetNull

Description

Sets a variable to null. The variable can be any datatype except for an array, structure, or autoinstantiated object.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

SetNull (*anyvariable*)

Argument	Description
anyvariable	The variable you want to set to null

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is null, SetNull returns null.

SetOption

Description

Sets an option for a camera device.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✗

Applies to

Camera objects

Syntax

Boolean *objectname*.SetOption (*Opt*, *iOptValue*)

Argument	Description
objectname	The name of the camera object for which you want to set an option.
Opt	A value of the CameraOption enumerated variable that specifies the name of the option that you want to want to inquire about. For a list of options, see GetOption.
iOptValue	An integer that specifies the value to which you want to set the option.

Return value

Integer. Returns 1 for success, and one of the following negative values if an error occurs:

-1	Unspecified error
-2	Supporting DLL not loaded
-3	Other initialization error
-5	Inconsistency in this object instance
-6	Call to the driver or device failed
-7	Unsupported option
-8	Value for option is out of range

Usage

Use the SetOption function to set the value of a specific option.

Examples

The following statements set the value of the CamOptWhiteBalance option to 3, which means fluorescent:

```
integer li_return
li_return = g_myCamera.SetOption(CamOptWhiteBalance, 3)
```

See also

CaptureImage
GetOption
IsReadyToCapture
Open
SetPreviewImageAttributes

SetOverlayPicture

Description

Puts an image in the control’s image list into an overlay image list.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

ListView and TreeView controls

Syntax

controlname.**SetOverlayPicture** (*overlayindex, imageindex*)

Argument	Description
controlname	The name of the ListView or TreeView control to which you want to add an overlay image.
overlayindex	The index number of the overlay picture in the overlay image list. The overlay image list is a 1-based array. *Overlayindex* must be 1 (for the first image), a previously designated index (replacing an image), or 1 greater than the current largest index (adding another image). SetOverlayPicture fails if you specify an index that creates gaps in the array.
imageindex	The index number of an image in the control's main image list. For ListViews, both the large and small pictures at that index become overlay images. The image is still available for use as an item's main image.

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs.

Usage

An overlay picture must have the same height and width as the picture it is used to overlay.

The color specified in the SetPictureMask property when the picture is inserted becomes transparent when the picture is used as an overlay, allowing part of the original image to be visible beneath the overlay.

The overlay list acts as a pointer back to the source image in the regular picture lists. If you delete an image that is also used in the overlay list, the displayed overlay pictures are affected too.

Examples

This example designates overlay images in a ListView control. The same picture is used for large and small images:

```
// Set up the overlay images
integer index
index = lv_1.AddLargePicture("shortcut.ico")
index = lv_1.AddSmallPicture("shortcut.ico")
lv_1.SetOverlayPicture(1, index)
index = lv_1.AddLargePicture("not.ico")
index = lv_1.AddSmallPicture("not.ico")
lv_1.SetOverlayPicture(2, index)

// Assign the second overlay image to the first item
listviewitem lvi
integer i
i = lv_1.GetItem(1, lvi)
lvi.OverlayPictureIndex = 2
i = lv_1.SetItem(1, lvi)
```

This example designates the first picture in the TreeView's main image list as the first overlay picture. The picture was added to the main image list on the TreeView's property sheet:

```
tv_list.SetOverlayPicture(1, 1)
```

This code in the TreeView's Clicked event assigns the overlay image to the clicked item:

```
treeviewitem tvi
tv_list.GetItem(handle, tvi)
tvi.OverlayPictureIndex = 1
tv_list.SetItem(handle, tvi)
```

SetParagraphSetting

Description: Sets the size of the indentation, left margin, or right margin of the paragraph containing the insertion point in a RichTextEdit control.

PocketBuilder	✗
PowerBuilder	✓

Applies to: RichTextEdit controls

Syntax: *rtecontrol*.**SetParagraphSetting** (*whichsetting*, *value*)

Return value: Integer. Returns 1 if it succeeds and -1 if an error occurs. If any argument is null, it returns null.

SetParm

Description: Reserved for future use. Sets the parameters to send to the MobiLink synchronization server.

Applies to: MLSynchronization, MLSync controls

Syntax: *SyncObject*.**SetParm** (*syncparm*)

Argument	Description
syncObject	The name of the synchronization object.
syncparm	A structure of type SyncParm containing property values that can be set as synchronization parameters.

Return value: Integer. Returns 1 for success and -1 for failure.

SetPicture

Description: Assigns an image stored in a blob to be the image in a Picture control.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to: Picture controls

Syntax

picturecontrol.**SetPicture** (*bimage*)

Argument	Description
picturecontrol	The name of a Picture control in which you want to set the bitmap.
bimage	A blob containing the new bitmap. *bimage* must be a valid picture in bitmap (BMP), Compuserve Graphics Interchange Format (GIF), Joint Photographic Experts Group (JPEG), run-length encoded (RLE), or Windows Metafile (WMF).

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is null, SetPicture returns null.

Usage

If you use FileRead to get the bitmap image from a file, remember that the FileRead function can read a maximum of 32765 characters at a time. To check the length of a file, call FileLength. If the file is over 32765 characters, you can call FileRead more than once and concatenate the return values.

Examples

These statements allow the user to select a file and then open the file and set the Picture control p_1 to the bitmap in the selected file:

```
integer fh, ret
blob Emp_pic
string txtname, named
string defext = "BMP"
string Filter = "bitmap Files (*.bmp), *.bmp"
ret = GetFileOpenName("Open Bitmap", txtname, &
       named, defext, filter)
IF ret = 1 THEN
       fh = FileOpen(txtname, StreamMode!)
       IF fh <> -1 THEN
          FileRead(fh, Emp_pic)
          FileClose(fh)
          p_1.SetPicture(Emp_pic)
       END IF
END IF
```

SetPointer

Description

Sets the mouse pointer to the specified shape.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Windows CE platforms
The pointer is an arrow by default. If you set the pointer to an hourglass in a desktop application, the pointer reverts to an arrow after the script is run. On a Windows CE device, you must explicitly call SetPointer a second time to reset the pointer.

Syntax

SetPointer (*type*)

Argument	Description
type	A value of the Pointer enumerated datatype indicating the type of pointer you want. Values are: Arrow! Cross! Beam! HourGlass! SizeNS! SizeNESW! SizeWE! SizeNWSE! UpArrow!

Return value

Pointer. Returns the enumerated type of the pointer it replaced so the script can restore it, if necessary. If *type* is null, SetPointer returns null.

Usage

Use SetPointer to display an hourglass at the beginning of a script when the script will take a long time to execute. The pointer remains set until you change it again in the script or the script terminates.

Restoring the arrow pointer
The pointer automatically changes back to an arrow when the script finishes executing. You do not have to change it back to an arrow.

In PocketBuilder's painters, you can specify the pointer shape that PocketBuilder displays when the user moves the pointer over a window, a control, or specific parts of a DataWindow object. The available shapes include the stock pointers listed above, as well as any custom cursor files you have.

Examples

This statement sets the pointer to the hourglass shape:

```
SetPointer(HourGlass!)
```

This example saves the old pointer and restores it when a long activity is completed:

```
pointer oldpointer // Declares a pointer variable
oldpointer = SetPointer(HourGlass!)
... // Performs some long activity
SetPointer(oldpointer)
```

SetPosition

Specifies the front-to-back position of a control in a window, a window, or an object within a DataWindow.

To	Use
Specify the front-to-back position of a control in a window, or specify that a window should always display on top of other windows	Syntax 1
Move an object in a DataWindow to another band or to specify its front-to-back position within a band	Syntax 2

Syntax 1 For positioning windows and controls in windows

Description

For controls in a window, specifies the position of a control in the front-to-back order within a window. For a window, specifies whether it always displays on top of other open windows.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

A control within a window or a window

Syntax

objectname.**SetPosition** (*position* {, *precedingobject* })

Argument	Description
objectname	The name of a control for which you want to specify a location in the front-to-back order within the window, or the name of a window for which you want to specify whether it always displays on top. *Objectname* cannot be a child window or a sheet.
position	A SetPosType enumerated datatype. The values you can specify depend on whether *objectname* is a control or a window. For controls, values are: • Behind! — Position *objectname* behind *precedingobject* in the order • ToTop! — Position *objectname* on top of all other controls • ToBottom! — Position *objectname* behind all other controls For windows, values are: • TopMost! — Always display *objectname* on top of all other open windows • NoTopMost! — Do not always display *objectname* on top of all other open windows
precedingobject (optional)	The name of the object you want to position *objectname* behind. *Precedingobject* is required if *position* is Behind!.

Return value

Integer. Returns 1 when it succeeds and -1 if an error occurs. If any argument's value is null, SetPosition returns null.

Usage

The front-to-back order for controls determines which control covers another when they overlap. If a control completely covers another control, the control that is in back becomes inaccessible to the user.

When you specify TopMost! for more than one window, the most recently executed SetPosition function controls which window displays on top.

Examples

This statement positions cb_two on top:

```
cb_two.SetPosition(ToTop!)
```

This statement positions cb_two behind cb_three:

```
cb_two.SetPosition(Behind!, cb_three)
```

This statement makes the window w_signon the topmost window:

```
w_signon.SetPosition(TopMost!)
```

This statement makes the window w_signon no longer necessarily the topmost window:

```
w_signon.SetPosition(NoTopMost!)
```

Syntax 2 For positioning objects within a DataWindow

Description

Moves an object within the DataWindow to another band or changes the front-to-back order of objects within a band.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

DataWindow controls and DataStores

Syntax

dwcontrol.**SetPosition** (*objectname*, *band*, *bringtofront*)

Argument	Description
dwcontrol	The name of the DataWindow control or DataStore containing the object.
objectname	The name of the object within the DataWindow that you want to move. You assign names to the DataWindow objects in the DataWindow painter.
band	The name of the band or layer in which you want to position *objectname*. Layer names are background and foreground. Band names are detail, header, footer, summary, header.#, and trailer.#. # is the group level number. Enter the empty string ("") if you do not want to change the band
bringtofront	A boolean indicating whether you want to bring *objectname* to the front within the band: • true — Bring it to the front • false — Do not bring it to the front

Return value

Integer. Returns 1 when it succeeds and -1 if an error occurs. If any argument's value is null, SetPosition returns null.

Examples

This statement moves *oval_red* in dw_rpt to the header and brings it to the front:

```
dw_rpt.SetPosition("oval_red", "header", TRUE)
```

This statement does not change the position of oval_red , but does bring it to the front:

```
dw_rpt.SetPosition("oval_red", "", TRUE)
```

This statement moves *oval_red* to the footer but does not bring it to the front:

```
dw_rpt.SetPosition("oval_red", "footer", FALSE)
```

SetPreviewImageAttributes

Description

Sets image attributes such as picture size and zoom value for previewing a picture.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✗

Applies to

Camera objects

Syntax

objectname.SetPreviewImageAttributes (*attrValue*)

Argument	Description
objectname	The name of the camera object for which you want to set preview attributes
attrValue	A CameraImageAttributes structure that contains the attributes to be set for the device

Return value

Integer. Returns 1 for success, and one of the following negative values if an error occurs:

-1	Unspecified error
-2	Supporting DLL not loaded
-3	Other initialization error
-5	Inconsistency in this object instance
-6	Call to the driver or device failed
-7	Unsupported option
-8	Value for option is out of range

Usage

You can set different attributes for previewing and capturing images. Typical preview values are 160 and 120 pixels for width and height and 2 for zoom. Image sizes available depend on the device, but usually fewer sizes are available for preview.

Examples

This example gets the attributes that are available for a device in an array of CameraImageAttributes structures and displays them to the user so that the user can select the set of attributes to be used for preview and capture:

```
CameraImageAttributes AllowedConfigs[]
g_myCam.GetAllowedImageAttributes(AllowedConfigs)

// Display choices to user and let user select
// a preview and capture configuration
...
// User chose 1 for preview, 3 for capture
g_myCam.SetPreviewImageAttributes(AllowedConfigs[1])
g_myCam.SetCaptureImageAttributes(AllowedConfigs[3])
```

See also

CaptureImage
GetOption
IsReadyToCapture
Open
SetCaptureImageAttributes
SetOption

SetProfileString

Description

Writes a value in a profile file for a PocketBuilder application.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

File format
On the desktop, SetProfileString writes to the file in the format, ANSI or Unicode, in which it was opened. On a deployment device, SetProfileString can write to files in Unicode format only. To write Unicode characters to an initialization file, open the file and save it as Unicode before calling SetProfileString.

Syntax

SetProfileString (*filename*, *section*, *key*, *value*)

Argument	Description
filename	A string whose value is the name of the profile file. For desktop applications, if you do not include the full path in *filename*, PocketBuilder searches the DOS path for *filename*.
section	A string whose value is the name of a group of related values in the profile file. If *section* does not exist in the file, PocketBuilder adds it.
key	A string whose value is the key in *section* for which you want to specify a value. If *key* does not exist in *section*, PocketBuilder adds it.
value	A string whose value is the value you want to specify for *key*.

Return value

Integer. Returns 1 when it succeeds and -1 if it fails because *filename* is not found or cannot be accessed. On a deployment device, SetProfileString returns -1 if the file is not in Unicode format.

Usage

A profile file consists of section labels, which are enclosed in square brackets, and keys, which are followed by an equal sign and a value. By changing the values assigned to the keys, you can specify custom settings for each installation of your application. When you are planning your own profile file, you select the section and key names and determine how the values are used.

For example, a profile file might contain information about the user. In the sample below, User Info is the section name and the other values are the keys. There is no space before and after the equal sign used in the keys or in the section label (if you use a section name such as Section=1):

```
[User Info]
Name="James Smith"
JobTitle="Window Washer"
SecurityClearance=9
Password=
```

Call SetProfileString to store configuration information, supplied by you or the user, in a profile file. You can call the functions ProfileInt and ProfileString to use that information to customize your PocketBuilder application during execution.

Accessing the profile file SetProfileString uses profile calls to write data to the profile file. Consequently it does not control when the profile file is written and closed. If you try to read data from the profile file immediately after calling SetProfileString, the file may still be open and you will receive incomplete or incorrect data.

To avoid this problem, you can use the PowerScript FileOpen, FileWrite, and FileClose functions to write data to the profile file instead of using SetProfileString. Or you can add some additional processing after the SetProfileString call so that the profile calls have time to complete before you try to read from the profile file.

Windows registry
SetProfileString can also be used to obtain configuration settings from the Windows system registry. For information on how to use the system registry, see the discussion of initialization files and the Windows registry in the *Resource Guide*.

Examples

This statement sets the keyword `Title` in section `Position` of file *C:\PROFILE.INI* to the string `MGR`:

```
SetProfileString("C:\PROFILE.INI", &
       "Position", "Title", "MGR")
```

See also

ProfileInt
ProfileString

SetRange

Description

Sets a duration for a progress bar control or sets the start and end position for a trackbar control.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Progress bar and trackbar controls

Syntax

controlname.**SetRange** (*startpos*, *endpos*)

Argument	Description
controlname	The name of the progress bar or trackbar
startpos	Integer indicating the initial position of the range
endpos	Integer indicating the terminal position of the range

Return value

Integer. Returns 1 if it succeeds and -1 if there is an error.

Usage

The default range for the progress bar controls is 0 to 100.

Examples

This statement sets a range of 1 to 10 for a progress bar control:

```
HProgressBar.SetRange ( 1, 10 )
```

See also

OffsetPos
SelectionRange
StepIt

SetRecordSet

Description

Sets an ADOResultSet object to obtain its data and metadata from a passed ADO Recordset.

PocketBuilder	✗
PowerBuilder	✓

Applies to

ADOResultSet objects

Syntax

adoresultset.**SetRecordSet** (*adorecordsetobject*)

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs.

SetRecurrence

Description

Sets a recurrence pattern for an appointment or task.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✗

Applies to

POOMAppointment, POOMTask objects

Syntax

Integer *objectname*.SetRecurrence (*pattern*)

Argument	Description
objectname	The name of the POOMAppointment or POOMTask object
pattern	The POOMRecurrence object with the pattern you want to set

Return value

Integer. Returns 1 for success and one of the following negative values if an error occurs:

-1	Unspecified error
-2	Cannot connect to the repository or a required internal subobject failed to connect to the repository
-3	Cannot log in to the repository
-4	Incorrect input argument
-5	Action cannot be performed
-6	The object identifier (OID) is not in the repository
-7	Feature is not implemented yet
-8	No matching entries found for the criteria

See also

ClearRecurrencePattern
GetRecurrence
SkipRecurrence

SetRedraw

Description

Controls the automatic redrawing of an object or control after each change to its properties.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Windows CE platforms
In PocketBuilder applications, SetRedraw (false) works only for the ListBox, DropDownListBox, and TreeView controls. SetRedraw (true) forces a repaint of all control types. This can lead to unexpected performance penalties in applications that you deploy to Pocket PC or Smartphone devices.

Applies to

Any object except a Menu

Syntax

objectname.**SetRedraw** (*boolean*)

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs. If *boolean* is null, SetRedraw returns null.

Usage

By default, PocketBuilder redraws a control after each change to properties that affect appearance. Use SetRedraw to turn off redrawing temporarily in order to avoid flicker and reduce redrawing time when you are making several changes to the properties of an object or control. If the window is not visible, SetRedraw fails.

Caution
If you turn redraw off, you must turn it on again. Otherwise, problems may result. In addition, if redraw is off and you change the Visible or Enabled property of an object in the window, the tabbing order may be affected.

Examples

This statement turns off redraw for lb_Location:

```
lb_Location.SetRedraw(FALSE)
```

If lb_Location is sorted (`lb_Location.Sorted = TRUE`), these statements use SetRedraw to avoid sorting and redrawing the list of lb_Location until all the new items have been added:

```
lb_Location.SetRedraw(FALSE)
lb_Location.AddItem("Atlanta")
lb_Location.AddItem("Boston")
lb_Location.AddItem("Washington")
lb_Location.SetRedraw(TRUE)
```

SetRegistrationCode

Description

Enables support for third-party software packages that require a registration code.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✗

Syntax

SetRegistrationCode (*idPackage*, *regvalue*)

Argument	Description
idPackage	An integer that references the software package on the Pocket PC device or emulator. Currently, the only recognized value is 1. This value references the FieldSoftware PrinterCE SDK.
regvalue	A string that sets the registration code supplied by the third-party software.

Return value

Integer. Returns 1 for success and -1 for failure. When an application is running on the desktop or in the development environment, SetRegistrationCode always returns -1.

Usage

Before you call print functions from an application that you deploy to a Pocket PC device or emulator, you can supply the registration code for third-party print software in the *regvalue* argument to the SetRegistrationCode function.

If you do not call SetRegistrationCode before you try to print from a deployed application, PocketBuilder assumes you are using an evaluation copy of the FieldSoftware PrinterCE SDK and attempts to make application print function calls using the evaluation software.

Examples

These statements in a Clicked event send the registration code for authorized use of the FieldSoftware PrinterCE SDK from a Pocket PC, then access this software to print the current page.

```
integer li_return
long ll_job

li_return = SetRegistrationCode(1,"555A55B555")
ll_job = PrintOpen("myprintjob")
li_return = PrintPage (li_job)
li_return = PrintClose (li_job)
```

See also

Print
Print method for DataWindows in the *DataWindow Reference*

SetRemote

Asks a DDE server application to accept data and store it in the specified location. There are two ways of calling SetRemote, depending on the type of DDE connection you have established.

To	Use
Make a single DDE request of a server application (a cold link)	Syntax 1
Make a DDE request of a server application when you have established a warm link by opening a channel	Syntax 2

Syntax 1 For single DDE requests

Description

Asks a DDE server application to accept data to be stored in the specified location without requiring an open channel. This syntax is appropriate when you will make only one or two requests of the server.

PocketBuilder	✗
PowerBuilder	✓

Syntax

SetRemote (*location*, *value*, *applname*, *topicname*)

Return value

Integer. Returns 1 if it succeeds and a negative integer if an error occurs.

Syntax 2 For DDE requests via an open channel

Description

Asks a DDE server application to accept data to be stored in the specified location when you have already established a warm link by opening a channel to the server. A warm link, with an open channel, is more efficient when you intend to make several DDE requests.

PocketBuilder	✗
PowerBuilder	✓

Syntax

SetRemote (*location*, *value*, *handle* {, *windowhandle* })

Return value

Integer. Returns 1 if it succeeds and a negative integer if an error occurs.

SetResultSet

Description — Populates a new ADOResultSet object with data passed in a ResultSet object.

PocketBuilder	✗
PowerBuilder	✓

Applies to — ADOResultSet objects

Syntax — *adoresultset*.**SetResultSet** (*resultsetobject*)

Return value — Integer. Returns 1 if it succeeds and -1 if an error occurs.

SetRingTone

Description — Specifies whether the receipt of an incoming call will play a sound file assigned by the PhoneCall object's RingTone property.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✗

Applies to — PhoneCall objects

Syntax — *objectname*.SetRingTone (*wavefile*)

Argument	Description
objectname	The name of the PhoneCall object.
wavefile	A read-only string that assigns a WAV sound file to the PhoneCall object's RingTone property.

Return value — Integer. Returns 1 for success and a negative value if an error occurs.

Usage — You can use an empty string for the *wavefile* argument to set the ring tone to the default ring (typically *\Windows\Rings\DefaultRing.wav*) stored on the device.

Examples — The following script for a Set Ring button sets the ring tone for the PhoneCall object.

```
String ls_sound
Integer li_return

ls_sound = "\Windows\Rings\MySpecialRing.wav"
```

```
if FileExists(ls_sound) = false then
   MessageBox("Error", "Sound file does not exist: " &
      + ls_sound)
else
      li_return = pcall_1.SetRingTone (ls_sound)
end if
```

See also

AcceptCall
AllowReceivingCalls
DropCall
MakeCall
SetHold
SetMute

SetRuntimeProperty

Description

Lets you set global properties and modify them at runtime.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✗

Syntax

integer **SetRuntimeProperty** (string *PropertyName*, boolean *fValue*)

Argument	Description
PropertyName	A read-only string for an internal runtime property that you want to modify at runtime. The properties you can modify are described in a table in the usage section for this function.
fvalue	A boolean for the value of the property that you want to modify at runtime.

Return value

Returns 0 for success and a negative number for failure.

Usage

The SetRuntimeProperty function has no effect on internal runtime properties in applications that you deploy to the desktop. It works only for applications running on a handheld device or emulator. The runtime properties you can set with this function are described in the following table:

Property	Description
AllowAutomaticSIPHandling	When you set this property to false, the PowerScript SIPOnFocus property is disabled for all controls. However, this does not prevent the application user from manually raising or lowering the SIP, or from changing its type. It also does not interfere with SIPUp and SIPDown events. By default, this property is set to true.
DrawFlatTabs	By default, this property is false and window tab controls have a classic 3-D look. You can display the tab controls with a traditional Pocket PC flat look at runtime by setting this property to true. The property does not affect tab control appearance in DataWindow objects, only in window objects. You should not try to modify the runtime appearance of tab controls when a tab control is already visible in an application window.
EnableExtraActivates	Windows operating systems typically provide fewer window activation and deactivation messages for mobile devices than for desktop computers. This is especially true for pop-up windows. Setting this property to true enables you to capture more window activation and deactivation messages on a mobile device without breaking legacy behavior. Unfortunately, it can also cause a window to receive two identical activation or deactivation messages. However, you can turn this property on or off at any time to fine tune the message capturing behavior.
MOPEnable	By setting this property to false, you disable MOP views or allow the end user of a PocketBuilder application to disable MOP views. This can be convenient if you want MOP views turned off only for a series of dialog boxes, after which, you can enable MOP views again by setting the MOPEnable property to true. When you turn off the MOPEnable property, the last values saved for X, Y, Width, and Height properties determine the positions and dimensions of windows and controls regardless of the screen orientation or size of the handheld device.

Property	Description
SIPOnFocusUp_AlternateTechnique	On some devices, the automatic SIP driver does not cause the SIP to display when text controls gain focus after a soft reboot. This property supplements the internal API of the device, ensuring that the preferred SIP position for the operating system is the up position. Typically you set this property only in the application Open event and the setting remains in effect for the lifetime of the application.
SIPOnFocusDown_AlternateTechnique	This property supplements the internal API on a handheld device, ensuring that the default SIP position for the operating system is the down position. Typically you set this property in the application Open event only if a particular device requires such an intervention. The property setting remains in effect for the lifetime of the application.

There is no equivalent “get” function for internal runtime property settings.

Examples

The following script sets the tabs on tab controls in application windows to a flat style:

```
integer li_return
li_return = SetRuntimeProperty (“DrawFlatTabs”, true)
```

SetScreenOrientation

Description

Sets the screen orientation of a device or emulator capable of screen rotation.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✗

Syntax

Integer SetScreenOrientation (*iOrientation*)

Argument	Description
iOrientation	A value corresponding to the screen orientation that you want to set. Values are: • **0** 0 degrees (the native orientation for the device) • **1** 90 degrees (right-handed landscape orientation) • **2** 180 degrees (upside down) • **4** 270 degrees (left-handed landscape orientation)

Return value

Returns 0 for success or a negative number for failure.

Usage

This function is supported on the Windows Mobile 2003 Second Edition platform. However, not all devices using this platform support screen rotation.

Examples

The following lines rotate the current screen to a right-handed landscape orientation:

```
integer iRotate, iRet
iRotate = 1
iRet = SetScreenOrientation( iRotate )
```

See also

GetScreenOrientation

SetSeriesStyle

Specifies the appearance of a series in a graph. There are several syntaxes, depending on what settings you want to change.

To	Use
Set the series' colors	Syntax 1
Set the line style and width	Syntax 2
Set the fill pattern or symbol for the series	Syntax 3
Specify that the series is an overlay	Syntax 4

Syntax 1 For setting a series' colors

Description

Specifies the colors of a series in a graph.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Graph controls in windows and user objects, and graphs in DataWindow controls

Syntax

controlname.**SetSeriesStyle** ({ *graphcontrol*, } *seriesname*, *colortype*, *color*)

Argument	Description
controlname	The name of the graph in which you want to set the color of a series, or the name of the DataWindow control containing the graph.
graphcontrol (DataWindow control only) (optional)	A string whose value is the name of the graph in the DataWindow control for which you want to set the color of a series.
seriesname	A string whose value is the name of the series for which you want to set the color.
colortype	A value of the grColorType enumerated datatype specifying the item for which you want to set the color. Values are: • Foreground! — Text color • Background! — Background color • LineColor! — Line color • Shade! — Shade (for graphics that are three-dimensional or have solid objects)
color	A long specifying the new color for *colortype*.

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is null, SetSeriesStyle returns null.

Usage

Data points in a series can have their own style settings. Settings made with SetDataStyle set the style of individual data points and override series settings.

The graph stores style information for properties that do not apply to the current graph type. For example, you can set the fill pattern in a two-dimensional line graph or the line style in a bar graph, but that fill pattern or line style will not be visible.

For a graph in a DataWindow, you can specify the appearance of a series in the graph before PocketBuilder draws the graph. To do so, define a user event for pbm_dwngraphcreate and call SetSeriesStyle in the script for that event. The event pbm_dwngraphcreate is triggered just before a graph is created in a DataWindow object.

Examples

This statement sets the text (foreground) color of the series named *Salary* in the graph gr_emp_data to black:

```
gr_emp_data.SetSeriesStyle("Salary", &
      Foreground!, 0)
```

This statement sets the background color of the series named *Salary* in the graph gr_depts in the DataWindow control dw_employees to black:

```
dw_employees.SetSeriesStyle("gr_depts", &
      "Salary", Background!, 0)
```

These statements in the Clicked event of the graph control gr_product_data coordinate line color between it and the graph gr_sales_data. The script stores the line color for the series under the mouse pointer in the graph gr_product_data in the variable *line_color*. Then it sets the line color for the series northeast in the graph gr_sales_data to that color:

```
string SeriesName
integer SeriesNbr, Series_Point
long line_color
grObjectType MouseHit

MouseHit = ObjectAtPointer(SeriesNbr,Series_Point)

IF MouseHit = TypeSeries! THEN
      SeriesName = &
         gr_product_data.SeriesName(SeriesNbr)

      gr_product_data.GetSeriesStyle(SeriesName, &
         LineColor!, line_color)

      gr_sales_data.SetSeriesStyle("Northeast", &
         LineColor!, line_color)
END IF
```

See also

GetDataStyle
GetSeriesStyle
SeriesName
SetDataStyle

Syntax 2 For lines in a graph

Description

Specifies the style and width of a series' lines in a graph.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Graph controls in windows and user objects, and graphs in DataWindow controls objects

Syntax

controlname.**SetSeriesStyle** ({ *graphcontrol*, } *seriesname*, *linestyle*, *linewidth*)

Argument	Description
controlname	The name of the graph in which you want to set the line style and width of a series, or the name of the DataWindow control containing the graph.
graphcontrol (DataWindow control only) (optional)	A string whose value is the name of the graph in the DataWindow control in which you want to set the line style and width.
seriesname	A string whose value is the name of the series for which you want to set the line style and width.
linestyle	A value of the LineStyle enumerated datatype. Values are: Continuous! Dash! DashDot! DashDotDot! Dot! Transparent!
linewidth	An integer specifying the width of the line in pixels.

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is null, SetSeriesStyle returns null.

Usage

Data points in a series can have their own style settings. Settings made with SetDataStyle set the style of individual data points and override series settings.

The graph stores style information for properties that do not apply to the current graph type. For example, you can set the fill pattern in a two-dimensional line graph or the line style in a bar graph, but that fill pattern or line style will not be visible.

For a graph in a DataWindow, you can specify the appearance of a series in the graph before PocketBuilder draws the graph. To do so, define a user event for pbm_dwngraphcreate and call SetSeriesStyle in the script for that event. The event pbm_dwngraphcreate is triggered just before a graph is created in a DataWindow object.

Examples

This statement sets the line style and width for the series named *Costs* in the graph gr_product_data:

```
gr_product_data.SetSeriesStyle("Costs", &
      Dot!, 5)
```

See also

GetDataStyle
GetSeriesStyle
SeriesName
SetDataStyle

Syntax 3 For the fill pattern and symbols in a graph

Description

Specifies the fill pattern and symbol for data markers in a series.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Graph controls in windows and user objects, and graphs in DataWindow controls

Syntax

controlname.**SetSeriesStyle** ({ *graphcontrol*, } *seriesname*, *enumvalue*)

Argument	Description
controlname	The name of the graph in which you want to set the appearance of a series, or the name of the DataWindow control containing the graph.
graphcontrol (DataWindow control only) (optional)	A string whose value is the name of the graph in the DataWindow control in which you want to set the appearance.

Argument	Description
seriesname	A string whose value is the name of the series in which you want to set the appearance.
enumvalue	A value of an enumerated datatype specifying an appearance setting for the series. To change the fill pattern, use a FillPattern enumerated datatype. FillPatter values are: Bdiagonal! (Lines from lower left to upper right) Diamond! Fdiagonal! (Lines from upper left to lower right) Horizontal! Solid! Square! Vertical! To change the symbol type, use a grSymbolType enumerated datatype. Values for grSymbolType are: NoSymbol! SymbolHollowBox! SymbolX! SymbolStar! SymbolHollowUpArrow! SymbolHollowCircle! SymbolHollowDiamond! SymbolSolidDownArrow! SymbolSolidUpArrow! SymbolSolidCircle! SymbolSolidDiamond! SymbolPlus! SymbolHollowDownArrow! SymbolSolidBox!

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is null, SetSeriesStyle returns null.

Usage

Data points in a series can have their own style settings. Settings made with SetDataStyle set the style of individual data points and override series settings.

The graph stores style information for properties that do not apply to the current graph type. For example, you can set the fill pattern in a two-dimensional line graph or the line style in a bar graph, but that fill pattern or line style will not be visible.

For a graph in a DataWindow, you can specify the appearance of a series in the graph before PocketBuilder draws the graph. To do so, define a user event for pbm_dwngraphcreate and call SetSeriesStyle in the script for that event. The event pbm_dwngraphcreate is triggered just before a graph is created in a DataWindow object.

Examples

This statement sets the symbol used for the series named *Costs* in the graph gr_product_data to a plus sign:

```
gr_product_data.SetSeriesStyle("Costs", &
       SymbolPlus!)
```

This statement sets the symbol used for the series named *Costs* in the graph gr_computers in the DataWindow control dw_equipment to X:

```
dw_equipment.SetSeriesStyle("gr_computers", &
       "Costs", SymbolX!)
```

See also

GetDataStyle
GetSeriesStyle
SeriesName
SetDataStyle

Syntax 4 For creating an overlay in a graph

Description

Specifies whether a series is an overlay, meaning that the series is represented by a line on top of another graph type.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Graph controls in windows and user objects, and graphs in DataWindow controls

Syntax

controlname.**SetSeriesStyle** ({ *graphcontrol*, } *seriesname*, *overlaystyle*)

Argument	Description
controlname	The name of the graph in which you want to set the overlay status of a series, or the name of the DataWindow control containing the graph.
graphcontrol (DataWindow control only) (optional)	A string whose value is the name of the graph in the DataWindow control in which you want to set the overlay status.

Argument	Description
seriesname	A string whose value is the name of the series whose overlay status you want to change.
overlaystyle	A boolean value indicating whether you want the series to be an overlay, meaning that the series is shown in front as a line. Set *overlaystyle* to true to make the specified series an overlay. Set it to false to remove the overlay setting.

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is null, SetSeriesStyle returns null.

Usage

For a graph in a DataWindow, you can specify the appearance of a series in the graph before PocketBuilder draws the graph. To do so, define a user event for pbm_dwngraphcreate and call SetSeriesStyle in the script for that event. The event pbm_dwngraphcreate is triggered just before a graph is created in a DataWindow object.

Examples

This statement sets the style of the series named *Costs* in the graph gr_product_data to overlay:

```
gr_product_data.SetSeriesStyle("Costs", TRUE)
```

These statements in the Clicked event of the DataWindow control dw_employees store the style of the series under the pointer in the graph gr_depts in the variable *style_type*. If the style of the series is overlay (true), the script changes the style to normal (false):

```
string SeriesName
integer SeriesNbr, Data_Point
boolean overlay_style
grObjectType MouseHit

MouseHit = dw_employees.ObjectAtPointer( &
      "gr_depts", SeriesNbr, Data_Point)

IF MouseHit = TypeSeries! THEN
      SeriesName = &

dw_employees.SeriesName("gr_depts",SeriesNbr)

      dw_employees.GetSeriesStyle("gr_depts", &
         SeriesName, overlay_style)

      IF overlay_style THEN &
         dw_employees.SetSeriesStyle("gr_depts", &
            SeriesName, FALSE)
END IF
```

See also	GetDataStyle GetSeriesStyle SeriesName SetDataStyle

SetSIPPreferredState

Description

Displays or hides the soft input panel (SIP) used on the Pocket PC.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✗

Syntax

SetSIPPreferredState (*hwnd*, *SIPState*)

Argument	Description
hwnd	A long that is the handle of the window or control that receives the SIP input.
SIPState	A value of the SIPState enumerated datatype that specifies the display state of the SIP. Values are: • SIPUp! – display the input panel • SIPDown! – hide the input panel after a timer expires • SIPForceDown! – hide the input panel immediately • SIPUnchanged! – ignore any pending requests to hide the input panel

Return value

Integer. Returns 0 for success and a negative value for failure. When running on the desktop or in the development environment, SetSIPPreferredState always returns 0.

Usage

Use SetSIPPreferredState to display the input panel when the application requires user input and hide it otherwise.

Do not call SetSIPPreferredState in an application until a window is active. The Open event for a window occurs too early to call this system function. If you need to call this function for a window, you can post a user event from the Open event and make the call from the posted event:

```
this.event POST ue_post_open()
```

If you specify SipUp!, any pending SipDown requests are ignored. When you specify SipDown!, a timer is set and the input panel is hidden when the timer expires. This prevents the input panel from flashing if another control requests SipUp!.

If you want the input panel hidden immediately and you are sure there will be no SipUp! requests, specify SipForceDown!.

If you specify SipDown! and then specify SipUnchanged! before the timer expires, the input panel remains in its current state.

Examples

These statements in the GetFocus event of a SingleLineEdit control display the input panel when the control gets focus:

```
integer li_return
li_return = SetSIPPreferredState(Handle(This),SIPUp!)
```

See also

IsSIPVisible
SetSIPType

SetSIPType

Description

Specifies the type of soft input panel (SIP) used on the Pocket PC.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✗

Syntax

SetSIPType (*SIPIMType*)

Argument	Description
SIPIMType	A value of the SIPIMType enumerated datatype that specifies the type of input panel you want to display. Values are: • SIPKeyboard! – standard keyboard • SIPJot! – letter recognizer • SIPBlock! – block recognizer • SIPWordLogic! – WordLogic keyboard • SIPTranscriber! – Microsoft Transcriber • SIPFitaly! – Fitaly keyboard for the Pocket PC

Return value

Integer. Returns 0 for success and a negative value for failure. When running on the desktop or in the development environment, SetSIPType always returns 0.

Usage

Use SetSIPType to set the input method (IM) used in the soft input panel on the Pocket PC. SetSIPType changes the global default SIP on the device and should therefore be used with caution. The IM requested must be installed on the Pocket PC. SetSIPType does *not* display or hide the input panel.

Examples

These statements set the Microsoft Transcriber as the default SIP type on the Pocket PC:

```
integer li_return
li_return = SetSIPType(SIPTranscriber!)
```

See also

GetSIPType
SetSIPPreferredState

SetSpacing

Description

Sets the line spacing for the selected paragraphs or the paragraph containing the insertion point in a RichTextEdit control.

PocketBuilder	✗
PowerBuilder	✓

Applies to

RichTextEdit controls

Syntax

rtename.**SetSpacing** (*spacing*)

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs.

SetState

Description

Sets the highlighted state of an item in a list box. SetState is only applicable to a list box control whose MultiSelect property is set to true.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✓

Applies to

ListBox and PictureListBox controls

Syntax

listboxname.**SetState** (*index*, *state*)

Argument	Description
listboxname	The name of the ListBox or PictureListBox in which you want to set the state (highlighted or not highlighted) for an item. The MultiSelect property for the control must be set to true.
index	The number of the item for which you want to set the state. Specify 0 to set the state of all the items in the ListBox.
state	A boolean value that determines the state of the item: • true — Selected • false — Not selected

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is null, SetState returns null.

Usage

When the MultiSelect property for the control is false, use SelectItem, instead of SetState, to select one item at a time.

Examples

This statement turns on the highlight for item 6 in lb_Actions:

```
lb_Actions.SetState(6, TRUE)
```

This statement deselects all items in lb_Actions:

```
lb_Actions.SetState(0, FALSE)
```

This statement turns off the highlight for item 6 in lb_Actions if it is selected and turns it on again if it is not selected:

```
IF lb_Actions.State(6) = 1 THEN
      lb_Actions.SetState(6, FALSE)
ELSE
      lb_Actions.SetState(6, TRUE)
END IF
```

See also

SelectItem
SetTop
State

SetSyncRegistryProperties

Description

Reserved for future use. Sets synchronization properties in the local machine registry.

Applies to

MLSynchronization, MLSync controls

Syntax *SyncObject*.**SetSyncRegistryProperties** ( )

Argument	Description
syncObject	The name of the synchronization object.

Return value Integer. Returns 1 for success and -1 for failure.

SetTextColor

Description Sets the color of selected text in a RichTextEdit control.

PocketBuilder	✗
PowerBuilder	✓

Applies to RichTextEdit controls

Syntax *rtename*.**SetTextColor** (*colornumber*)

Return value Integer. Returns 1 if it succeeds and -1 if an error occurs.

SetTextStyle

Description Specifies the text formatting for selected text in a RichTextEdit control. You can make the text bold, underlined, italic, and struck out. You can also make it either a subscript or superscript.

PocketBuilder	✗
PowerBuilder	✓

Applies to RichTextEdit controls

Syntax *rtename*.**SetTextStyle** (*bold*, *underline*, *subscript*, *superscript*, *italic*, *strikeout*)

Return value Integer. Returns 1 if it succeeds and -1 if an error occurs.

SetToolbar

Description

Specifies the alignment, visibility, and title for the specified toolbar.

PocketBuilder	✗
PowerBuilder	✓

Applies to

MDI frame and sheet windows

Syntax

window.**SetToolbar** (*toolbarindex*, *visible* {, *alignment* {, *floatingtitle* } })

Return value

Integer. Returns 1 if it succeeds. SetToolbar returns -1 if there is no toolbar for the index you specify or if an error occurs. If any argument's value is null, returns null.

SetToolbarPos

Sets the position of the specified toolbar.

To set	Use
Docking position of a docked toolbar	Syntax 1
Coordinates and size of a floating toolbar	Syntax 2

Syntax 1 For docked toolbars

Description

Sets the position of a docked toolbar.

PocketBuilder	✗
PowerBuilder	✓

Applies to

MDI frame and sheet windows

Syntax

window.**SetToolbarPos** (*toolbarindex*, *dockrow*, *offset*, *insert*)

Return value

Integer. Returns 1 if it succeeds. SetToolbarPos returns -1 if there is no toolbar for the index you specify or if an error occurs. If any argument's value is null, returns null.

Syntax 2 For floating toolbars

Description

Sets the position and size of a floating toolbar.

PocketBuilder	✗
PowerBuilder	✓

Applies to

MDI frame and sheet windows

Syntax

window.**SetToolbarPos** (*toolbarindex*, *x*, *y*, *width*, *height*)

Return value

Integer. Returns 1 if it succeeds. SetToolbarPos returns -1 if there is no toolbar for the index you specify or if an error occurs. If any argument's value is null, SetToolbarPos returns null.

SetTop

Description

Scrolls a list box control so that the specified item is the first visible item.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

ListBox and PictureListBox controls

Syntax

listboxname.**SetTop** (*index*)

Argument	Description
listboxname	The name of the ListBox or PictureListBox that you want to scroll
index	The number of the item you want to become the first visible item

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is null, SetTop returns null.

Examples

This statement scrolls item 6 in lb_Actions to the top of the ListBox so that it is the first visible item:

```
lb_Actions.SetTop(6)
```

The following statement scrolls the currently selected item in lb_Actions to the top of the list of items:

```
lb_Actions.SetTop(lb_Actions.SelectedIndex())
```

See also

SetFocus
SetState

SetTraceFileName

Description

Specifies the name of the trace file PocketBuilder will analyze when the BuildModel function is called.

PocketBuilder on Desktop	✓
PocketBuilder on Pocket PC	✗
PocketBuilder on Smartphone	✗
PowerBuilder	✓

Applies to

Profiling and TraceTree objects

Syntax

***instancename*.SetTraceFileName** (*tracefilename*)

Argument	Description
instancename	Instance name of the Profiling or TraceTree object
tracefilename	A string that identifies the name of the trace file PocketBuilder will analyze

Return value

ErrorReturn. Returns one of the following values:

- Success!—The function succeeded
- FileOpenError!—The file could not be opened
- FileInvalidFormatError!—The trace file is not in the correct format
- ModelExistsError!—A model has already been built

If an error occurs, the name is not set.

Usage

Use this function to specify the trace file PocketBuilder should analyze with the BuildModel function. You call the SetTraceFileName function before calling the BuildModel function.

Examples

This example provides the name of the trace file for which a performance analysis model is to be built:

```
Profiling lpro_model
String ls_line
```

```
lpro_model = CREATE Profiling

lpro_model.SetTraceFileName (filename)
ls_line = "CollectionTime = " + &
      String(lpro_model.CollectionTime ) + "~r~n" &
         + "Num Activities = " &
         + String(lpro_model.NumberOfActivities) +
"~r~n"

lpro_model.BuildModel()
...
```

See also

BuildModel

SetTransPool

Description

Sets up a pool of database transactions for an application. SetTransPool allows you to minimize the overhead associated with database connections and also limit the total number of database connections permitted.

PocketBuilder	✗
PowerBuilder	✓

Applies to

Application object

Syntax

applicationname.**SetTransPool** (*minimum*, *maximum*, *timeout*)

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs.

SharedObjectDirectory

Description
Retrieves the list of objects that have been registered for sharing.

PocketBuilder	✗
PowerBuilder	✓

Syntax
SharedObjectDirectory (*instancenames* {, *classnames* })

Return value
ErrorReturn. Returns one of the following values:

- Success! — The function succeeded
- FeatureNotSupportedError! — This function is not supported on this platform

SharedObjectGet

Description
Gets a reference to a shared object instance.

PocketBuilder	✗
PowerBuilder	✓

Syntax
SharedObjectGet (*instancename* , *objectinstance*)

Return value
ErrorReturn. Returns one of the following values:

- Success! — The function succeeded
- SharedObjectCreateInstanceError! — The local reference to the shared object could not be created
- SharedObjectNotExistsError! — The instance name has not been registered

SharedObjectRegister

Description
Registers a user object so that it can be shared.

PocketBuilder	✗
PowerBuilder	✓

Syntax **SharedObjectRegister** (*classname* , *instancename*)

Return value ErrorReturn. Returns one of the following values:

- Success! — The function succeeded
- SharedObjectExistsError! — The instance name has already been used
- SharedObjectCreateInstanceError! — The object could not be created
- SharedObjectCreatePBSessionError! — The shared object session could not be created

SharedObjectUnregister

Description Unregisters a user object that was previously registered.

PocketBuilder	✗
PowerBuilder	✓

Syntax **SharedObjectUnregister** (*instancename*)

Return value ErrorReturn. Returns one of the following values:

- Success! — The function succeeded
- SharedObjectNotExistsError! — The instance name has not been registered

Show

Description Makes an object or control visible, if it is hidden. If the object is already visible, Show brings it to the top.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to Any object

Syntax

objectname.**Show** ( )

Argument	Description
objectname	The name of the object or control you want to make visible (show)

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs. If *objectname* is null, Show returns null.

Usage

If the specified object is a window that is not open, an execution error occurs.

You cannot use Show to show a drop-down or cascading menu, or any menu that has an MDI frame window as its parent window.

Equivalent syntax You can set the object's Visible property instead of calling Show:

objectname.Visible = true

This statement:

```
m_status.m_options.Visible = TRUE
```

is equivalent to:

```
m_status.m_options.Show()
```

Examples

This statement makes visible the menu selection called m_options on the menu m_status:

```
m_status.m_options.Show()
```

This statement makes the child window w_child visible:

```
w_child.Show()
```

See also

Hide

ShowHeadFoot

Description

Displays the panels for editing the header and footer in a RichTextEdit control or hides the panels and returns to editing the main text.

PocketBuilder	✗
PowerBuilder	✓

Applies to

RichTextEdit controls and DataWindow controls with the RichTextEdit presentation style

Syntax *rtename*.**ShowHeadFoot** (*editheadfoot*)

Return value Integer. Returns 1 if it succeeds and -1 if an error occurs.

ShowHelp

Description Provides access to a Microsoft Windows-based Help system or to compiled HTML Help files that you have created for your PowerBuilder application. When you call ShowHelp, PowerBuilder starts the Help executable and displays the Help file you specify.

PocketBuilder	✗
PowerBuilder	✓

Syntax **ShowHelp** (*helpfile*, *helpcommand* {, *typeid* })

Return value Integer. Returns 1 if it succeeds and -1 if an error occurs. ShowHelp returns -1 if you specify *typeid* when *helpcommand* is Finder! or Index!. If any argument's value is null, ShowHelp returns null.

ShowPopupHelp

Description Displays pop-up help for the specified control.

PocketBuilder	✗
PowerBuilder	✓

Applies to Any control

Syntax **ShowPopupHelp** (*helpfile*, *control*, *contextid*)

Return value Integer. Returns 1 if the function succeeds and -1 if an error occurs.

Sign

Description

Reports whether a number is negative, zero, or positive.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

Sign (*n*)

Argument	Description
n	The number for which you want to find out the sign

Return value

Integer. Returns a number (-1, 0, or 1) indicating the sign of *n*. If *n* is null, Sign returns null.

Examples

This statement returns 1 (the number is positive):

```
Sign(5)
```

This statement returns 0 (zero has no sign):

```
Sign(0)
```

This statement returns -1 (the number is negative):

```
Sign(-5)
```

See also

Sign method for DataWindows in the *DataWindow Reference*

SignalError

Description

Causes a SystemError event at the application level.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

SignalError ({ *number* }, { *text* })

Argument	Description
number (optional)	The integer (stored in the number property of the Error object) to be used in the message object
text (optional)	The string (stored in the text property of the Error object) to be used in the message object

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs. The return value is usually not used.

Usage

During development you can use SignalError to test error-processing scripts.You can call PopulateError to populate the Error object and call SignalError without arguments. You can examine how the SystemError event script handles the forced error. If you pass the optional *number* and *text* arguments to SignalError, it populates all the fields in the Error object and then triggers a SystemError event.

In an application, SignalError can also be useful. For example, if a user error is so severe that you do not want the application to continue, you can set values in the Error object, including your own error number, and call SignalError. You need to include code in the SystemError event script to recognize and handle the error you have created.If there is no script for the SystemError event, the SignalError function does nothing.

For the execution-time error numbers assigned to the Number property of the Error object when an application error occurs, see the *Users Guide*.

Examples

These statements set values in the Error object and then trigger a SystemError event so the error processing for these values can be tested:

```
int error_number
string error_text
Error.Number = 1010
Error.Text = "Salary must be a positive number."
Error.Windowmenu = "w_emp"

error_number = Error.Number
error_text = Error.Text

SignalError(error_number, error_text)
```

See also

PopulateError

Sin

Description

Calculates the sine of an angle.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

Sin (*n*)

Argument	Description
n	The angle (in radians) for which you want the sine

Return value

Double. Returns the sine of *n*. If *n* is null, Sin returns null.

Examples

This statement returns .8414709848078965:

```
Sin(1)
```

This statement returns 0:

```
Sin(0)
```

This statement returns 0:

```
Sin(Pi(1))
```

See also

ASin
Cos
Pi
Tan
Sin method for DataWindows in the *DataWindow Reference*

SkipRecurrence

Description

Moves to the next occurrence in a recurring task.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✗

Applies to

POOMTask objects

Syntax

Integer *objectname*.SkipRecurrence ( )

Argument	Description
objectname	The name of the POOMTask object

Return value

Integer. Returns 1 for success and one of the following negative values if an error occurs:

-1	Unspecified error
-2	Cannot connect to the repository or a required internal subobject failed to connect to the repository
-3	Cannot log in to the repository
-4	Incorrect input argument
-5	Action cannot be performed
-6	The object identifier (OID) is not in the repository
-7	Feature is not implemented yet
-8	No matching entries found for the criteria

See also

ClearRecurrencePattern
GetRecurrence
SetRecurrence

Sleep

Description

Causes the application to pause for a specified time.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

Sleep (*seconds*)

Argument	Description
seconds	Long for the number of seconds you want the application to pause

Return value

Integer. Returns 1 if the function succeeds and -1 if an error occurs.

Examples

This example pauses the application for 5 seconds:

```
Sleep ( 5 )
```

SoftTrigger

Sets or retrieves the state of the soft trigger feature of a scanner. The soft trigger is a common feature that enables a scanner laser through software, rather than through a trigger button.

To	Use
Retrieve the state of the soft trigger feature	Syntax 1
Set the state of the soft trigger feature	Syntax 2

Syntax 1 For retrieving the soft trigger state

Description

Retrieves the soft trigger state of a scanner.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✗

Applies to

BarcodeScanner

Syntax

Boolean *scannerobject*.SoftTrigger ( )

Argument	Description
scannerobject	The name of the bar code scanner object

Return value

Boolean. Value are:

- **True** The soft trigger feature is on.
- **False** The soft trigger feature is off.

Examples

The following example returns the state of the soft trigger feature of the scanner asscociated with the BarcodeScanner object l_scanner:

```
lb_softstate = l_scanner.SoftTrigger()
```

Syntax 2 For setting the soft trigger state

Description

Sets the soft trigger state of a scanner.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✗

Applies to

BarcodeScanner

Syntax

Integer *scannerobject*.SoftTrigger (*newState*)

Argument	Description
scannerobject	The name of the bar code scanner object
newState	Boolean value that is used to set the soft trigger state

Return value

Integer. Returns 1 for success or one of the following negative values if an error occurs:

- **-1** Unspecified error
- **-2** Supporting DLL not loaded error
- **-3** Initialization error other than DLL not loaded
- **-4** Error in the passed in arguments
- **-5** Something in the object instance is inconsistent
- **-6** Call to the driver failed
- **-7** Error opening the specific scan device
- **-8** Error in the internal buffer allocation
- **-9** Incorrect scan state for the requested action (typically benign)
- **-10** Low level device error
- **-11** Read is already pending (typically benign)
- **-12** Read is cancelled (typically benign)
- **-13** Timeout period expired on the read (typically benign)
- **-14** Error creating the asynchronous read from the message sink
- **-100** Feature not implemented

Examples

The following example turns off the soft trigger feature of the scanner asscociated with the BarcodeScanner object l_scanner:

```
li_rtn = l_scanner.SoftTrigger(false)
```

Sort

Sorts rows in a DataWindow control, DataStore, or child DataWindow, or items in a TreeView or ListView control.

For syntax for DataWindows and DataStores, see the Sort method for DataWindows in the *DataWindow Reference* or the online Help.

To sort	Use
Items in a TreeView	Syntax 1
Items in a ListView	Syntax 2

Syntax 1 For TreeView controls

Description
Sorts the children of an item in a TreeView control.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to
TreeView controls

Syntax
treeviewname.**Sort** (*itemhandle* , *sorttype*)

Argument	Description
treeviewname	The name of the TreeView control in which you want to sort items.
itemhandle	The item for which you want to sort its children.
sorttype	The sort method you want to use. Valid values are: Ascending! Descending! UserDefinedSort!

Return value
Integer. Returns 1 if it succeeds and -1 if it fails.

Usage
The Sort function only sorts the immediate level beneath the specified item. If you want to sort multiple levels, use SortAll. If you specify UserDefinedSort! as your *sorttype*, define your sort criteria in the Sort event of the TreeView control. The Sort function cannot sort level 1 of a TreeView. However, level 1 is sorted automatically when the TreeView's SortType property calls for sorting.

Examples
This example sorts the children of the current TreeView item:

```
long ll_tvi
```

```
ll_tvi = tv_foo.FindItem(CurrentTreeItem! , 0)
tv_foo.SetRedraw(false)
tv_foo.Sort(ll_tvi , Ascending!)
tv_foo.SetRedraw(true)
```

See also SortAll

Syntax 2 For ListView controls

Description Sorts items in ListView controls.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to ListView controls

Syntax *listviewname*.**Sort** (*sorttype*, { *column* })

Argument	Description
listviewname	The ListView in which you want to sort items.
sorttype	The method you want to use when you sort the ListView items. Values are: Ascending! Descending! Unsorted! UserDefinedSort!
column (optional)	The number of the column by which you wish to sort the ListView items.

Return value Integer. Returns 1 if it succeeds and -1 if it fails.

Usage The default sort is alphanumeric.

If you do not specify a column to sort, the first column is sorted.

Examples This example sorts the items in column three of a ListView:

```
lv_list.SetRedraw(false)
lv_list.Sort(Ascending! , 3)
lv_list.SetRedraw(true)
```

See also SortAll

SortAll

Description

Sorts all the levels below an item in the TreeView item hierarchy.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

TreeView controls

Syntax

treeviewname.**SortAll** (*itemhandle*, *sorttype*)

Argument	Description
treeviewname	The TreeView control in which you want to sort the subsequent levels in an item's hierarchy.
itemhandle	The item for which you want to sort all the levels below it.
sorttype	The sort method you want to use. Values are: Ascending! Descending! Unsorted! UserDefinedSort!

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs.

Usage

If you specify UserDefinedSort! as your *sorttype*, define your sort criteria in the Sort event of the TreeView control.

The SortAll function cannot sort level 1 of a TreeView. However, level 1 is sorted automatically when the TreeView's SortType property calls for sorting.

Examples

This example sorts the subsequent levels recursively under the current TreeView item:

```
long ll_tvi

//Find the current treeitem
ll_tvi = tv_list.FindItem(CurrentTreeItem! , 0)

//Sort all children
tv_list.SortAll(ll_tvi , Ascending!)
```

This example recursively sorts the entire TreeView control:

```
long ll_tvi

//Find the root treeitem
ll_tvi = tv_list.FindItem(RootTreeItem! , 0)
```

```
//Sort all children
tv_list.SortAll(ll_tvi , Ascending!)
```

See also

Sort

Space

Description

Builds a string of the specified length whose value consists of spaces.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

Space (*n*)

Argument	Description
n	A long whose value is the length of the string you want filled with spaces. The maximum value is 2,147,483,647, which is the maximum size for strings.

Return value

String. Returns a string filled with *n* spaces if it succeeds and the empty string ("") if an error occurs. If *n* is null, Space returns null.

Examples

This statement puts a string whose value is four spaces in *Name*:

```
string Name
Name = Space(4)
```

This statement assigns 40 spaces to the string *Name*:

```
string Name
Name = Space(40)
```

See also

Fill
Space method for DataWindows in the *DataWindow Reference*

Sqrt

Description

Calculates the square root of a number.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

Sqrt (*n*)

Argument	Description
n	The number for which you want the square root

Return value

Double. Returns the square root of *n*. If *n* is null, Sqrt returns null.

Usage

Sqrt(*n*) is the same as *n*^.5.

Taking the square root of a negative number causes an execution error.

Examples

This statement returns 1.414213562373095:

```
Sqrt(2)
```

This statement results in an error at execution time:

```
Sqrt(-2)
```

See also

Sqrt method for DataWindows in the *DataWindow Reference*

Start

Start has two basic syntaxes.

To	Use
Execute a pipeline object	Syntax 1
Activate a timing object	Syntax 2

Syntax 1 For executing pipeline objects

Description

Executes a pipeline object, which transfers data from the source to the destination as specified by the SQL query in the pipeline object. This pipeline object is a property of a user object inherited from the pipeline system object.

PocketBuilder	✗
PowerBuilder	✓

Applies to

Pipeline objects

Syntax

pipelineobject.**Start** (*sourcetrans*, *destinationtrans*, *errorobject* {, *arg1*, *arg2*,..., *argn* })

Return value

Integer. Returns 1 if it succeeds and a negative number if an error occurs.

Syntax 2 For activating timing objects

Description

Activates a timing object causing a Timer event to occur repeatedly at the specified interval.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Timing objects

Syntax

timingobject.**Start** (*interval*)

Return value

Integer. Returns 1 if it succeeds and -1 if the timer is already running, the interval specified is invalid, or there are no system timers available.

StartHotLink

Description

Establishes a hot link with a DDE server application so that PowerBuilder is notified immediately of any changes in the specified data. When the data changes in the server application, it triggers a HotLinkAlarm event in the current application.

PocketBuilder	✗
PowerBuilder	✓

Syntax	**StartHotLink** (*location*, *applname*, *topic*)
Return value	Integer. Returns 1 if it succeeds. If an error occurs, StartHotLink returns a negative integer.

StartServerDDE

Description: Establishes your application as a DDE server. You specify the DDE name, topic, and items that you support.

PocketBuilder	✗
PowerBuilder	✓

Syntax	**StartServerDDE** ({ *windowname*, } *applname*, *topic* {, *item* })
Return value	Integer. Returns 1 if it succeeds. If an error occurs, StartServerDDE returns -1, meaning the your application is already started as a server. If any argument's value is null, StartServerDDE returns null.

State

Description: Determines whether an item in a ListBox control is highlighted.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to	ListBox and PictureListBox controls
Syntax	*listboxname*.**State** (*index*)

Argument	Description
listboxname	The name of the ListBox or PictureListBox in which you want to obtain the state (highlighted or not highlighted) of the item identified by *index*
index	The number of the item for which you want to obtain the state

Return value: Integer. Returns 1 if the item in *listboxname* identified by *index* is highlighted and 0 if it is not. If the index does not point to a valid item number, State returns -1. If any argument's value is null, State returns null.

Usage

The State and SetState functions are meant for a ListBox that allows multiple selections (its MultiSelect property is true). To find all of a list's selected items, loop through the list, checking the state of each item.

The SelectedItem and SelectItem functions are meant for single-selection ListBox controls. SelectedItem reports the selection directly with no need for looping. In a multiple-selection ListBox control, SelectedItem reports the first selected item only.

When you know the index of an item, you can use the Text function to get the item's text.

Examples

If item 3 in lb_Contact is selected (highlighted), then this example sets *li_Item* to 1:

```
integer li_Item
li_Item = lb_Contact.State(3)
```

The following statements obtain the text of all the selected items in a ListBox that allows the user to select more than one item. The MessageBox function displays each item as it is found. You could include other processing that created an array or list of the selected values:

```
integer li_ItemTotal, li_ItemCount

// Get the number of items in the ListBox.
li_ItemTotal = lb_contact.TotalItems( )

// Loop through all the items.
FOR li_ItemCount = 1 to li_ItemTotal
   // Is the item selected? If so, display the text
   IF lb_Contact.State(li_ItemCount) = 1 THEN &
   MessageBox("Selected Item", &
   lb_Contact.text(li_ItemCount))
NEXT
```

This statement executes some statements if item 3 in the ListBox lb_Contact is highlighted:

```
IF lb_Contact.State(3) = 1 THEN ...
```

See also

SelectedItem
SetState

Status

Description

Returns the current status of the scanner.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✗

Applies to

BarcodeScanner

Syntax

Integer *scannerobject*.ScannerStatus ( )

Argument	Description
scannerobject	The name of the bar code scanner object

Return value

Integer. Returns one of the following values:

- **11** Scanner not enabled
- **12** Scanner is enabled, but no reads are pending
- **13** One or more reads are pending, waiting for trigger event
- **14** Beam is on and acquiring data
- **15** Beam is on for aiming
- **16** Beam is off and waiting for firmware buffers to recover

Usage

Typically you might call the Status function during exception processing to determine why the scanner error occured or a RetrieveData call failed.

Examples

The following example returns the status of the scanner asscociated with the BarcodeScanner object l_scanner:

```
li_rtn = l_scanner.Status()
```

See also

RetrieveData

StepIt

Description

Increments the current position in a progress bar control by the value specified in the SetStep property of the control.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Progress bar controls

Syntax

control.**StepIt** ( )

Argument	Description
control	The name of the progress bar

Return value

Integer. Returns 1 if it succeeds and -1 if there is an error.

Usage

StepIt causes the position in a progress bar to wrap if the value of the SetStep takes the current position out of range. For example, if the SetStep value is 40, the current position 80, and the range is set from 0 to 100, the position on the redrawn progress bar after you call StepIt is 20.

The SetStep property can have a negative value. The default value for SetStep is 10.

Examples

This statement adds the SetStep increment to a progress bar control:

```
HProgressBar.StepIt ( )
```

See also

SetRange

Stop

Description

Deactivates a timing object.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Timing objects

Syntax

timingobject.**Stop** ( )

Return value

Integer. Returns 1 if it succeeds and -1 if the timer is not running or could not be stopped.

StopHotLink

Description

Terminates a hot link with a DDE server application.

PocketBuilder	✗
PowerBuilder	✓

Caution
All arguments must match the arguments in an earlier StartHotLink call.

Syntax

StopHotLink (*location*, *applname*, *topic*)

Return value

Integer. Returns 1 if it succeeds. If an error occurs, StopHotLink returns a negative integer.

StopServerDDE

Description

Causes your application to stop acting as a DDE server application. *Any subsequent requests* from a DDE client application fail.

PocketBuilder	✗
PowerBuilder	✓

Syntax

StopServerDDE ({ *windowname*, } *applname*, *topic*)

Return value

Integer. Returns 1 if it succeeds. If an error occurs, StopServerDDE returns -1, meaning the DDE server was not started. If any argument's value is null, StopServerDDE returns null.

String

String has two syntaxes.

To	Use
Format data as a string according to a specified display format mask	Syntax 1
Convert a blob to a string	Syntax 2

Syntax 1 For formatting data

Description

Formats data, such as time or date values, according to a format mask. You can convert and format date, DateTime, numeric, and time data. You can also apply a display format to a string.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

String (*data*, { *format* })

Argument	Description
data	The data you want returned as a string with the specified formatting. *Data* can have a date, DateTime, numeric, time, or string datatype. *Data* can also be an Any variable containing one of these datatypes.
format (optional)	A string whose value is the display masks you want to use to format the data. The masks consists of formatting information specific to the datatype of *data*. If *data* is type string, *format* is required. The format can consist of more than one mask, depending on the datatype of *data*. Each mask is separated by a semicolon. (For details on each datatype, see Usage).

Return value

String. Returns *data* in the specified format if it succeeds and the empty string ("") if the datatype of *data* does not match the type of display mask specified, *format* is not a valid mask, or *data* is an incompatible datatype.

Usage

For date, DateTime, numeric, and time data, PocketBuilder uses the system's default format for the returned string if you do not specify a format. For numeric data, the default format is the [General] format.

For string data, a display format mask is required. (Otherwise, the function would have nothing to do.)

The format can consist of one or more masks:

- Formats for date, DateTime, string, and time data can include one or two masks. The first mask is the format for the data; the second mask is the format for a null value.
- Formats for numeric data can have up to four masks. A format with a single mask handles both positive and negative data. If there are additional masks, the first mask is for positive values, and the additional masks are for negative, zero, and null values.

To display additional characters as part of the mask for a decimal value, you must precede each character with a backslash. For example, to display a decimal number with two digits of precision preceded by four asterisks, you must type a backslash before each asterisk:

```
dec{2} amount
string = ls_result
amount = 123456.32
ls_result = string(amount,"\*\*\*\*0.00")
```

The resulting string is `****123456.32`.

For more information on specifying display formats, see the *Users Guide*. Note that, although a format can include color specifications, the colors are ignored when you use String in PowerScript. Colors appear only for display formats specified in the DataWindow painter.

If the display format does not match the datatype, PocketBuilder tries to apply the mask, which can produce unpredictable results.

Times and dates from a DataWindow control
When you call GetItemTime or GetItemString as an argument for the String function and do not specify a display format, the value is formatted as a DateTime value. This statement returns a string like "2/26/03 00:00:00":

```
String(dw_1.GetItemTime(1, "start_date"))
```

International deployment When you use String to format a date and the month is displayed as text (for example, the display format includes "mmm"), the month is in the language of the runtime DLLs available when the application is run. If you have installed localized runtime files in the development environment or on a user's machine, then on that machine, the month in the resulting string is in the language of the localized files.

For information about the localized runtime files, which are available in French, German, Italian, Spanish, Dutch, Danish, Norwegian, and Swedish, see the chapter on internationalization in *Application Techniques*.

Message object You can also use String to extract a string from the Message object after calling TriggerEvent or PostEvent. For more information, see the TriggerEvent or PostEvent functions.

Examples

This statement applies a display format to a date value and returns `Jan 31, 2002`:

```
String(2002-01-31, "mmm dd, yyyy")
```

This example applies a format to the value in *order_date* and sets *date1* to `6-11-02`:

```
Date order_date = 2002-06-11
string date1
date1 = String(order_date,"m-d-yy")
```

This example includes a format for a null date value so that when *order_date* is null, *date1* is set to none:

```
Date order_date = 2002-06-11
string date1
SetNull(order_date)
date1 = String(order_date, "m-d-yy;'none'")
```

This statement applies a format to a DateTime value and returns `Jan 31, 2001 6 hrs and 8 min`:

```
String(DateTime(2001-01-31, 06:08:00), &
   'mmm dd, yyyy h "hrs and" m "min"')
```

This example builds a DateTime value from the system date and time using the Today and Now functions. The String function applies formatting and sets the text of sle_date to that value, for example, `6-11-02 8:06 pm`:

```
DateTime sys_datetime
string datetime1
sys_datetime = DateTime(Today(), Now())
sle_date.text = String(sys_datetime, &
   "m-d-yy h:mm am/pm;'none'")
```

This statement applies a format to a numeric value and returns `$5.00`:

```
String(5,"$#,##0.00")
```

The statements in the following table set the string variable *string1*:

PowerScript code	Result

`integer nbr = 123`
`string string1`
`string1 = String(nbr,"0000;(000);****;empty")`
	0123

`integer nbr = -123`
`string string1`
`string1 = String(nbr,"000;(000);****;empty")`
	123

PowerScript code	Result

`integer nbr = 0`
`string string1`
`string1 = **String**(nbr,"0000;(000);****;empty")`
	****

`integer nbr`
`string string1`
`SetNull(nbr)`
`string1 = **String**(nbr,"0000;(000);****;empty")`
	“empty”

The statements in the following table format string data, assigning a characer in the source string to each `@` and inserting other characters in the format at the appropriate positions:

Statement	Result
String("ABC", "@-@-@")	A-B-C
String("ABC", "@*@")	A*B
String("ABC", "@@@")	ABC
String("ABC", " ")	blank space

The statements in the following table apply display formats to time data:

Statement	Result
String(06:08:02,'h "hrs and" m "min"')	6 hrs and 8 min
String(20:06:04,"hh:mm:ss am/pm")	08:06:04 pm
String(08:06:04,"h:mm:ss am/pm")	08:06:04 pm

PocketBuilder allows you to use the String function to convert numeric data to hexadecimal formatting. The statements in the following table apply hexadecimal formatting to the decimal value 12:

Statement	Result
String(12,"hex")	C
String(12,"hex2")	0C
String(12,"hex4")	000C
String(12,"hex8")	0000000C

See also

String method for DataWindows in the *DataWindow Reference*

Syntax 2 For blobs

Description

Converts data in a blob to a string value. If the blob's value is not text data, String attempts to interpret the data as characters.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

String (*blob*)

Argument	Description
blob	The blob whose value you want returned as a string. *Blob* can also be an Any variable containing a blob.

Return value

String. Returns the value of *blob* as a string if it succeeds and the empty string ("") if it fails. It the blob does not contain string data, String interprets the data as characters, if possible, and returns a string. If *blob* is null, String returns null.

Usage

You can also use String to extract a string from the Message object after calling TriggerEvent or PostEvent. For more information, see the TriggerEvent or PostEvent functions.

Examples

This example converts the blob instance variable *ib_sblob*, which contains string data, to a string and stores the result in *sstr*:

```
string sstr
sstr = String(ib_sblob)
```

This example stores today's date and test status information in the blob *bb*. *Pos1* and *pos2* store the beginning and end of the status text in the blob. Finally, BlobMid extracts a "sub-blob" that String converts to a string. Sle_status displays the returned status text:

```
blob{100} bb
long pos1, pos2
string test_status
date test_date

test_date = Today()
IF DayName(test_date) = "Wednesday" THEN &
   test_status = "Coolant Test"
IF DayName(test_date) = "Thursday" THEN &
   test_status = "Emissions Test"

// Store data in the blob
pos1 = BlobEdit( bb, 1, test_date)
```

```
pos2 = BlobEdit( bb, pos1, test_status )

... // Some processing

// Extract the status stored in bb and display it
sle_status.text = String( &
   BlobMid(bb, pos1, pos2 - pos1))
```

See also

String method for DataWindows in the *DataWindow Reference*

String_To_Object

Description

Gets an object reference based on a passed string.

This function is used by PowerBuilder clients connecting to EAServer.

PocketBuilder	✗
PowerBuilder	✓

Applies to

JaguarORB objects

Syntax

jaguarorb.**String_To_Object** (*objstring* , *object*)

Return value

Long. Returns 0 if it succeeds and a negative number if an error occurs.

SuspendTransaction

Description

Suspends the EAServer transaction associated with the calling thread.

PocketBuilder	✗
PowerBuilder	✓

Applies to

CORBACurrent objects

Syntax

CORBACurrent.**SuspendTransaction** ( )

Return value

Unsigned long. Returns a handle that refers to the transaction associated with the thread or 0 if an error occurs.

Synchronize

Starts synchronization between a remote and consolidated database. The syntax you use depends on whether you include command line parameters with the dbmlsync synchronization call.

To start synchronization	Use
Without including command line parameters	Syntax 1
With command line parameters that you include in the synchroniztion call	Syntax 2

Syntax 1 For synchronization without parameters

Description

Reserved for future use. Starts synchronization between a remote and consolidated database.

Applies to

MLSynchronization, MLSync controls

Syntax

SyncObject.**Synchronize** ( )

Argument	Description
syncObject	The name of the synchronization object.

Return value

Integer. Returns 1 for success and -1 for failure. Any other return value is an error code from dbmlsync.

Syntax 2 For synchronization with parameters

Description

Reserved for future use. Starts dbmlsync synchronization with command line parameters that are passed from the values of a syncparm structure.

Applies to

MLSync controls

Syntax

SyncObject.**Synchronize** (*cmdstring*)

Argument	Description
syncObject	The name of the synchronization object.
cmdstring	A read-only string containing command line arguments for a synchronization call.

Return value

Integer. Returns 1 for success and any other value for failure.

SyntaxFromSQL

Description

Generates DataWindow source code based on a SQL SELECT statement.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Transaction objects

Syntax

transaction.**SyntaxFromSQL** (*sqlselect*, *presentation*, *err*)

Argument	Description
transaction	The name of a connected transaction object.
sqlselect	A string whose value is a valid SQL SELECT statement.
presentation	A string whose value is the default presentation style you want for the DataWindow. The simple format is: Style(Type=*presentationstyle*) Values for *presentationstyle* correspond to the styles in the New DataWindow dialog box in the DataWindow painter. Keywords are: (Default) Tabular Grid Form (for freeform) Graph Group Label Nup The Usage section lists the keywords you can use in *presentation*.
err	A string variable to which PocketBuilder will assign any error messages that occur.

Return value

String. Returns the empty string ("") if an error occurs. If SyntaxFromSQL fails, *err* may contain error messages if warnings or soft errors occur (for example, a syntax error). If any argument's value is null, SyntaxFromSQL returns null.

Usage

To create a DataWindow object, you can pass the source code returned by SyntaxFromSQL directly to the Create function.

Table owner in the SQL statement If the value of the LogID property of the Transaction object is not the owner of the table being accessed in the SQL statement for the SyntaxFromSQL function, then the table name in the SQL SELECT statement must be qualified with the owner name.

The *presentation* string can also specify object keywords followed by properties and values to customize the DataWindow. You can specify the style of a column, the entire DataWindow, areas of the DataWindow, and text in the DataWindow. The object keywords are:

Column	Group	Text
DataWindow	Style	Title

A full presentation string has the format:

"Style (Type=*value property*=*value* ...)
  DataWindow (*property*=*value* ...)
  Column (*property*=*value* ...)
  Group *groupby_colnum1 Fby_colnum2* ... *property* ...)
  Text *property*=*value* ...)
  Title ('*titlestring*')"

The checklists in the DataWindow object properties chapter in the *DataWindow Reference* identify the properties that you can use for each object keyword.

If a database column has extended attributes with font information, then font information you specify in the SyntaxFromSQL presentation string is ignored.

Examples

The following statements display the DataWindow source for a tabular DataWindow object generated by the SyntaxFromSQL function in a MultiLineEdit. If errors occur, PocketBuilder fills the string *ERRORS* with any error messages that are generated:

```
string ERRORS, sql_syntax

sql_syntax = "SELECT emp_data.emp_id," &
   + "emp_data.emp_name FROM emp_data " &
   + "WHERE emp_data.emp_salary >45000"

mle_sql.text = &
   SQLCA.SyntaxFromSQL(sql_syntax, "", ERRORS)
```

The following statements create a grid DataWindow dw_1 from the DataWindow source generated in the SyntaxFromSQL function. If errors occur, the string *ERRORS* contains any error messages that are generated, which are displayed to the user in a message box. Note that you need to call SetTransObject with SQLCA as its argument before you can call the Retrieve function:

```
string ERRORS, sql_syntax
string presentation_str, dwsyntax_str

sql_syntax = "SELECT emp_data.emp_id,"&
   + "emp_data.emp_name FROM emp_data "&
   + "WHERE emp_data.emp_salary > 45000"

presentation_str = "style(type=grid)"

dwsyntax_str = SQLCA.SyntaxFromSQL(sql_syntax, &
   presentation_str, ERRORS)

IF Len(ERRORS) > 0 THEN
   MessageBox("Caution", &
   "SyntaxFromSQL caused these errors: " + ERRORS)
   RETURN
END IF

dw_1.Create( dwsyntax_str, ERRORS)

IF Len(ERRORS) > 0 THEN
   MessageBox("Caution", &
      "Create cause these errors: " + ERRORS)
   RETURN
END IF
```

See also

Create method for DataWindows in the *DataWindow Reference*

SystemRoutine

Description

Provides the routine node representing the system root in a performance analysis model.

PocketBuilder on Desktop	✓
PocketBuilder on Pocket PC	✗
PocketBuilder on Smartphone	✗
PowerBuilder	✓

Applies to

Profiling object

Syntax

instancename.**SystemRoutine** (*theroutine*)

Argument	Description
instancename	Instance name of the Profiling object.
theroutine	A value of type ProfileRoutine containing the routine node representing the system root. This argument is passed by reference.

Return value

ErrorReturn. Returns one of the following values:

- Success!—The function succeeded
- ModelNotExistsError!—The function failed because no model exists

Usage

Use this function to extract the routine node representing the system root in a performance analysis model. You must have previously created the performance analysis model from a trace file using the BuildModel function. The routine node is defined as a ProfileRoutine object and provides the time spent in the routine, any called routines, the number of times each routine was called, and the class to which the routine belongs.

Examples

This example provides the routine that represents the system root in a performance analysis model:

```
Profiling lpro_model
ProfileRoutine lprort_routine

lpro_model.BuildModel()
lpro_model.SystemRoutine(lprort_routine)
...
```

See also

BuildModel

TabPostEvent

Description

Posts the specified event for each tab page in a Tab control, adding them to the end of the event queues for the tab page user objects.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✓

Applies to

Tab controls

Syntax

tabcontrolname.**TabPostEvent** (*event* {, *word*, *long* })

Argument	Description
tabcontrolname	The name of the Tab control for which you want to post events for its tab page user objects.
event	A value of the TrigEvent enumerated datatype that identifies a PocketBuilder event (for example, Clicked!, Modified!, or DoubleClicked!) or a string whose value is the name of an event. The event must be a valid event for a tab page user object in *tabcontrolname* and a script must exist for the event in *tabcontrolname*.
word (optional)	A long value to be stored in the WordParm property of the system's Message object. If you want to specify a value for *long*, but not *word*, enter 0. (For cross-platform compatibility, WordParm and LongParm are both longs).
long (optional)	A long value or a string that you want to store in the LongParm property of the system's Message object. When you specify a string, a pointer to the string is stored in the LongParm property, which you can access with the String function (see Usage for PostEvent).

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs, if the event is not a valid event for the tab page user object, or if a script does not exist for the event.

Examples

Suppose tab_address contains several tab pages inherited from uo_list and uo_list has a user event called ue_display. This statement posts the event ue_display for each the tab pages in tab_address:

```
tab_address.TabPostEvent("ue_display")
```

See also

TabTriggerEvent

TabTriggerEvent

Description

Triggers the specified event for each tab page in a Tab control, which executes the scripts immediately in the index order of the tab pages.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✓

Applies to

Tab controls

Syntax

tabcontrolname.**TabTriggerEvent** (*event* {, *word*, *long* })

Argument	Description
tabcontrolname	The name of the Tab control for which you want to trigger events for its tab page user objects.
event	A value of the TrigEvent enumerated datatype that identifies a PocketBuilder event (for example, Clicked!, Modified!, or DoubleClicked!) or a string whose value is the name of an event. The event must be a valid event for a tab page user object in *tabcontrolname* and a script must exist for the event in *tabcontrolname*.
word (optional)	A long value to be stored in the WordParm property of the system's Message object. If you want to specify a value for *long*, but not *word*, enter 0. (For cross-platform compatibility, WordParm and LongParm are both longs).
long (optional)	A long value or a string that you want to store in the LongParm property of the system's Message object. When you specify a string, a pointer to the string is stored in the LongParm property, which you can access with the String function (see Usage for TriggerEvent).

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs, if the event is not a valid event for the tab page user object, or if a script does not exist for the event.

Examples

Suppose tab_address contains several tab pages inherited from uo_list and uo_list has a user event called ue_display. This statement executes immediately the script for ue_display for each the tab pages in tab_address:

```
tab_address.TabTriggerEvent("ue_display")
```

See also

TabPostEvent

Tan

Description

Calculates the tangent of an angle.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

Tan (*n*)

Argument	Description
n	The angle (in radians) for which you want the tangent

Return value

Double. Returns the tangent of *n*. An execution error occurs if *n* is not valid. If *n* is null, Tan returns null.

Examples

Both these statements return 0:

```
Tan(0)
Tan(Pi(1))
```

This statement returns 1.55741:

```
Tan(1)
```

See also

ATan
Cos
Pi
Sin
Tan method for DataWindows in the *DataWindow Reference*

Text

Description

Obtains the text of an item in a ListBox control.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

ListBox, DropDownListBox, PictureListBox, and DropDownPictureListBox controls

Syntax

listboxname.**Text** (*index*)

Argument	Description
listboxname	The name of the ListBox control in which you want the text of an item
index	The number of the item for which you want the text

Return value

String. Returns the text of the item in *listboxname* identified by *index*. If the index does not point to a valid item number, Text returns the empty string (""). If any argument's value is null, Text returns null.

Examples

Assume the ListBox lb_Cities contains:

Atlanta
Boston
Chicago
Denver

Then these statements store the text of item 3, which is Chicago, in *current_city*:

```
string current_city
current_city = lb_Cities.Text(3)
```

See also

FindItem
SelectedItem
SelectedText

TextLine

Description

Obtains the text of the line that contains the insertion point. TextLine works for controls that can contain multiple lines.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

DataWindow, EditMask, MultiLineEdit, and RichTextEdit controls

Syntax

editname.**TextLine** ( )

Argument	Description
editname	The name of the control in which you want the text on the line that contains the insertion point

Return value

String. Returns the text on the line with the insertion point in *editname*. If an error occurs, TextLine returns the empty string (""). If *editname* is null, TextLine returns null.

Usage

If *editname* is a DataWindow control, then TextLine reports information about the edit control over the current row and column.

Examples

In the MultiLineEdit mle_state, if the insertion point is on line 4 and its text is North Carolina, then this example sets *linetext* to North Carolina:

```
string linetext
linetext = mle_state.TextLine()
```

If the insertion point is on a line whose text is Y in the MultiLineEdit mle_contact, then some processing takes place:

```
IF mle_contact.TextLine() = "Y" THEN ...
```

See also

SelectedItem
SelectTextLine

Time

Converts DateTime, string, or numeric data to data of type time. It also extracts a time value from a blob. You can use one of three syntaxes, depending on the datatype of the source data.

To	Use
Extract the time from DateTime data, or to extract a time stored in a blob	Syntax 1
Convert a string to a time	Syntax 2
Combine numbers for hours, minutes, and seconds into a time value	Syntax 3

Syntax 1 For DateTime and blob values

Description

Extracts a time value from a DateTime value or a blob.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

Time (*datetime*)

Argument	Description
datetime	A DateTime value or a blob in which the first value is a time or DateTime value. The rest of the contents of the blob is ignored. *Datetime* can also be an Any variable containing a DateTime or blob.

Return value

Time. Returns the time in *datetime* as a time. If *datetime* does not contain a valid time or is an incompatible datatype, Time returns 00:00:00.000000. If *datetime* is null, Time returns null.

Examples

After *StartDateTime* has been retrieved from the database, this example sets *StartTime* equal to the time in *StartDateTime*:

```
DateTime StartDateTime
time StartTime
...
StartTime = Time(StartDateTime)
```

Suppose that the value of a blob variable ib_blob contains a DateTime value beginning at byte 32. The following statement extracts the time from the value:

```
time lt_time
lt_time = Time(BlobMid(ib_blob, 32))
```

See also

Time method for DataWindows in the *DataWindow Reference*

Syntax 2 For strings

Description

Converts a string containing a valid time into a time value.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

Time (*string*)

Argument	Description
string	A string whose value is a valid time (such as 8am or 10:25) that you want returned as a time. Only the hour is required; you do not have to include the minutes, seconds, or microseconds of the time or am or pm. The default value is 00 for minutes and seconds and 000000 for microseconds. PocketBuilder determines whether the time is am or pm based on a 24-hour clock. *String* can also be an Any variable containing a string or blob.

Return value

Time. Returns the time in *string* as a time. If string does not contain a valid time or is an incompatible datatype, Time returns 00:00:00.000000. If *string* is null, Time returns null.

Usage

Valid times can include any combination of hours (00 to 23), minutes (00 to 59), seconds (00 to 59), and microseconds (0 to 999999).

Examples

These statements set *What_Time* to null:

```
Time What_Time
string null_string

SetNull(null_string)
What_Time = Time(null_string)
```

This statement returns a time value for 45 seconds before midnight (23:59:15), which is specified as a string:

```
Time("23:59:15")
```

This statement converts the text in the SingleLineEdit sle_Time_Received to a time value:

```
Time(sle_Time_Received.Text)
```

See also

Time method for DataWindows in the *DataWindow Reference*

Syntax 3 For integers

Description

Combines integers representing hours, minutes, seconds, and microseconds into a time value.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

Time (*hour*, *minute*, *second* {, *microsecond* })

Argument	Description
hour	The integer for the hour (00 to 23) of the time
minute	The integer for the minutes (00 to 59) of the time
second	The integer for the seconds (0 to 59) of the time
microsecond (optional)	The integer for the microseconds (0 to 32767) of the time (note that the range of values supported for this argument is less than the total range of values possible for a microsecond)

Return value

Time. Returns the time as a time datatype and 00:00:00 if the value in any argument is not valid (out of the specified range of values). If any argument is null, Time returns null.

Examples

These statements set *What_Time* to a time value with microseconds, and display the resulting time as a string in st_1. The default display format does not include microseconds, so the String function specifies a display format with microseconds. Leading zeros are appended to the string value for microseconds:

```
Time What_Time
What_Time = Time(10, 15, 45, 234)
st_1.Text = String(What_Time, "hh:mm:ss:ffffff")
```

The time in the string variable is set to 10:15:45:000234.

These statements set *What_Time* to 10:15:45:

```
Time What_Time
What_Time = Time(10, 15, 45)
```

See also

Time method for DataWindows in the *DataWindow Reference*

Timer

Description

Causes a Timer event in a window to occur repeatedly at the specified interval. When you call Timer, it starts a timer. When the interval is over, PocketBuilder triggers the Timer event and resets the timer.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

Timer (*interval* {, *windowname* })

Argument	Description
interval	The number of seconds that you want between Timer events. interval can be a whole number or fraction greater than 0 and less than or equal to 4,294,967 seconds. If *interval* is 0, Timer turns off the timer so that it no longer triggers Timer events.
windowname (optional)	The window in which you want the timer event to be triggered. The window must be an open window. If you do not specify a window, the Timer event occurs in the current window.

Return value

Integer. Returns 1 if succeeds and -1 if an error occurs. If any argument's value is null, Timer returns null.

Usage

Do not call the Timer function in the Timer event. The timer gets reset automatically and the Timer event retrigger sat the interval that has already been established. Call the Timer function in another event's script when you want to stop the timer or change the interval.

Examples

This statement triggers a Timer event every two seconds in the active window:

```
Timer(2)
```

This statement stops the triggering of the Timer event in the active window:

```
Timer(0)
```

These statements trigger a Timer event every half second in the window w_Train:

```
Open(w_Train)
Timer(0.5, w_Train)
```

This example causes the current time to be displayed in a StaticText control in a window. Calling Timer in the window's Open event script starts the timer. The script for the Timer event refreshes the displayed time.

In the window's Open event script, the following code displays the time initially and starts the timer:

```
st_time.Text = String(Now(), "hh:mm")
Timer(60)
```

In the window's Timer event, which is triggered every minute, this code displays the current time in the StaticText st_time:

```
st_time.Text = String(Now(), "hh:mm")
```

See also

Idle

ToAnsi

Description

Converts a character string to an ANSI blob.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

ToAnsi (*string*)

Argument	Description
string	A character string you want to convert to an ANSI blob

Return value

Blob. Returns an ANSI blob if it succeeds and an empty blob if it fails.

Usage

In PocketBuilder, the ToAnsi function converts a Unicode character string to an ANSI blob.

Unicode file format
Unicode files sometimes have two extra bytes at the start of the file to indicate that they are Unicode files.If the two bytes are missing, PocketBuilder assumes "little endian" format. If you are opening a Unicode file in stream mode, skip the first two bytes if they are present

Examples

This example converts a string into an ANSI blob using the ToAnsi function and then writes the blob to a file.

```
integer  li_filenum
blob     lblb_text
string   ls_native
```

```
ls_native = "Sample text in native format"
lblb_text = ToAnsi(ls_native)

li_filenum = FileOpen("ansi.txt", StreamMode!, &
      Write!, LockWrite!, Replace!)

FileWrite(li_filenum, lblb_text)
FileClose(li_filenum)
```

See also

Blob
FromAnsi
FromUnicode
ToUnicode

Today

Description

Obtains the system date and, in some cases, the system time.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

Today ( )

Return value

Date. Returns the current system date.

Usage

Although the datatype of the Today function is date, it can also return the current time. This occurs when Today is used as an argument for another function and that argument allows different datatypes.

For example, if you call Today as an argument to the String function, String returns both the date and time when you use a date-plus-time display format. A second example: if you call Today as an argument for the SetItem function and the datatype of the target column is DateTime, both the date and time are assigned to the DataWindow.

Examples

This statement returns the current system date:

```
Today()
```

This statement executes some statements when the current system date is before April 15, 2003:

```
IF Today() < 2003-04-15 THEN ...
```

This statement displays the current date in the StaticText st_date in the corner of a window:

```
st_date.Text = String(Today(), "m/d/yy")
```

This statement displays the current date and time in the StaticText st_date:

```
st_date.Text = String(Today(), "m/d/yy hh:mm")
```

See also

Now
Today method for DataWindows in the *DataWindow Reference*

TodaySave

Description

Saves changes to the Today item in the device registry and refreshes the Today screen.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✗

Applies to

Application object

Syntax

Integer *appname*.TodaySave ( )

Argument	Description
controlname	The name of the application for which you want to save and display changes to the Today screen

Return value

Integer. Returns 1 for success, -1 if there is an error.

Usage

Use the TodaySave function to permanently save any property changes to the custom Today item. When you call this function, any changes you make to the custom item's display text, to its order in the Today screen, or to its display or run application, are saved to the registry. After the user restarts the device, the properties of the custom item are initialized to the changed values in the device registry.

Examples

The following example updates the registry for a Today item display text associated with the SyncDisplay application:

```
SyncDisplay.TodayDisplayText="Sync Update Count is " &
 + string(counter)
li_rtn = SyncDisplay.TodaySave()
```

Top

Description

Obtains the index number of the first visible item in a ListBox control. Top lets you to find out how the user has scrolled the list.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

ListBox and PictureListBox controls

Syntax

listboxname.**Top** ( )

Argument	Description
listboxname	The name of the ListBox or PictureListBox in which you want the index of the first visible item in the list

Return value

Integer. Returns the index of the first visible item in *listboxname*. Top returns -1 if an error occurs. If *listboxname* is null, Top returns null.

Usage

The index of a list item is its position in the full list of items, regardless of how many are currently visible in the control.

Examples

If item 15 has been scrolled to the top of the list in lb_Contacts, then this example sets *Num* to 15:

```
integer Num
Num = lb_Contacts.Top()
```

If the user has not scrolled the list in lb_Contacts, then *Num* is set to 1:

```
integer Num
Num = lb_Contacts.Top()
```

If the item at the top of the list in lb_Contacts is not the currently selected item, the following statements scroll the currently selected item to the top:

```
integer Num
```

```
Num = lb_Contacts.SelectedIndex()
IF lb_Contacts.Top() <> Num THEN &
     lb_contacts.SetTop(Num)
```

See also

SelectedIndex
SetTop

TotalColumns

Description

Finds the number of columns in a ListView control.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

ListView controls

Syntax

listviewname.**TotalColumns** ( )

Argument	Description
listviewname	The name of the ListView control for which you want to find the number of columns

Return value

Integer. Returns the number of columns if it succeeds and -1 if an error occurs.

Usage

Use when the ListView control is set to report view.

Examples

This example displays the number of columns in a ListView report view in a SingleLineEdit:

```
int li_cols
li_cols = lv_list.TotalColumns()
sle_info.text = "Total columns = " + string(li_cols)
```

See also

TotalItems
TotalSelected

TotalItems

Description

Determines the total number of items in a ListBox control.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

ListBox, DropDownListBox, PictureListBox, DropDownPictureListBox, and ListView controls

Syntax

listcontrolname.**TotalItems** ( )

Argument	Description
listcontrolname	The name of the control in which you want the total number of items

Return value

Integer. Returns the total number of items in *listcontrolname*. If *listcontrolname* contains no items, TotalItems returns 0. If an error occurs, it returns -1. If *listcontrolname* is null, TotalItems returns null.

Examples

If lb_Actions contains a total of five items, this example sets Total to 5:

```
integer Total
Total = lbx_Actions.TotalItems()
```

This FOR loop is executed for each item in lb_Actions:

```
integer Total, n
Total = lb_Actions.TotalItems()
FOR n = 1 to Total
... // Some processing
NEXT
```

See also

TotalSelected

TotalSelected

Description

Determines the number of items in a ListBox control that are selected.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

ListBox, PictureListBox, and ListView controls

Syntax

listcontrolname.**TotalSelected** ( )

Argument	Description
listcontrolname	The name of the control in which you want the number of items that are selected

Return value

Integer. Returns the number of items in *listcontrolname* that are selected. If no items in *listcontrolname* are selected, TotalSelected returns 0. If an error occurs, it returns -1. If *listcontrolname* is null, TotalSelected returns null.

Usage

TotalSelected works only if the MultiSelect property of *listcontrolname* is TRUE.

Examples

If three items are selected in lb_Actions, this example sets *SelectedTotal* to 3:

```
integer SelectedTotal
SelectedTotal = lb_Actions.TotalSelected()
```

These statements in the SelectionChanged event of lb_Actions display a MessageBox if the user tries to select more than three items:

```
IF lb_Actions.TotalSelected() > 3 THEN
      MessageBox("Warning", &
         "You can only select 3 items!")
ELSE
... // Some processing
END IF
```

See also

TotalItems

ToUnicode

Description

Converts a character string to a Unicode blob.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

ToUnicode (*string*)

Argument	Description
string	A character string you want to convert to a Unicode blob

Return value

Blob. Returns a Unicode blob if it succeeds and an empty blob if it fails.

Usage

In PocketBuilder, the ToUnicode function converts a Unicode character string to a blob and has the same result as Blob(*string*). In PowerBuilder, the ToUnicode function converts an ANSI character string to a Unicode blob.

Unicode file format
Unicode files sometimes have two extra bytes at the start of the file to indicate that they are Unicode files.

Examples

This example illustrates the use of the ToUnicode function to convert a string entered in a MultilineEdit control into a Unicode blob:

```
blob  lblb_text
string  ls_native

ls_native = mle_entry.Text
lblb_text = ToUnicode(ls_native)
```

See also

FromAnsi
FromUnicode
ToAnsi

TraceBegin

Description

Inserts an activity type value in the trace file indicating that logging has begun and then starts logging all the enabled application trace activities. Before calling TraceBegin, you must have opened the trace file using the TraceOpen function.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

TraceBegin (*identifier*)

Argument	Description
identifier	A read-only string, logged to the trace file, used to identify a tracing block. If *identifier* is null, an empty string is placed in the trace file.

Return value

ErrorReturn. Returns one of the following values:

- Success!—The function succeeded
- FileNotOpenError!—TraceOpen has not been called yet
- TraceStartedError!—TraceBegin has already been called

Usage

The TraceBegin call inserts an activity type value of ActBegin! in the trace file to indicate that logging has begun and then begins logging all the application activities you have selected for tracing.

TraceBegin can only be called following a TraceOpen call. And all activities to be logged must be enabled using the TraceEnableActivity function before calling TraceBegin.

If you want to generate a trace file for an entire application run, you typically include the TraceBegin function in your application’s open script. If you want to generate a trace file for only a portion of the application run, you typically include the TraceBegin function in the script that initiates the functionality on which you’re trying to collect data.

You can use the *identifier* argument to identify the tracing blocks within a trace file. A tracing block represents the data logged between calls to TraceBegin and TraceEnd. There may be multiple tracing blocks within a single trace file if you are tracing more than one portion of the application run.

Examples

This example opens a trace file with the name you entered in a single line edit box and a timer kind selected from a drop-down list. It then begins logging the enabled activities for the first block of code to be traced:

```
TimerKind ltk_kind

CHOOSE CASE ddlb_timestamp.text
CASE "None"
      ltk_kind = TimerNone!
CASE "Clock"
      ltk_kind = Clock!
CASE "Process"
      ltk_kind = Process!
CASE "Thread"
      ltk_kind = Thread!
END CHOOSE

TraceOpen(sle_filename.text,ltk_kind)

TraceEnableActivity(ActESQL!)
TraceEnableActivity(ActGarbageCollect!)
TraceEnableActivity(ActObjectCreate!)
TraceEnableActivity(ActObjectDestroy!)

TraceBegin("Trace_block_1")
```

See also
TraceOpen
TraceEnableActivity
TraceEnd

TraceClose

Description
Closes the trace file.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax
TraceClose ( )

Return value
ErrorReturn. Returns one of the following values:

- Success!—The function succeeded
- FileNotOpenError!—TraceOpen has not been called yet
- FileCloseError!—The log file is full

Usage
TraceClose closes the trace file. If you have not already called TraceEnd, TraceClose will call that function before proceeding with its processing.

You typically include the TraceClose function in your application's Close script.

Examples
This example stops logging of application trace activities and then closes the open trace file:

```
TraceEnd()
TraceClose()
```

See also
TraceBegin
TraceEnd
TraceOpen

TraceDisableActivity

Description

Disables logging of the specified trace activity.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

TraceDisableActivity (*activity*)

Argument	Description
activity	A value of the enumerated datatype TraceActivity that identifies the activity for which logging should be disabled. Values are: • ActError!—Occurrences of system errors and warnings • ActESQL!—Embedded SQL statement entry and exit • ActGarbageCollect!—Start and finish of garbage collection • ActLine!—Routine line hits • ActObjectCreate!—Object creation entry and exit • ActObjectDestroy!—Object destruction entry and exit • ActProfile!—Abbreviation for the ActRoutine!, ActESQL!, ActObjectCreate!, ActObjectDestroy!, and ActGarbageCollect! values • ActRoutine!—Routine entry and exit (if this value is disabled, ActLine! is automatically disabled) • ActTrace!—Abbreviation for all activities except ActLine! • ActUser!—Occurrences of an activity you selected

Return value

ErrorReturn. Returns one of the following values:

- Success!—The function succeeded
- FileNotOpenError!—TraceOpen has not been called yet
- TraceStartedError!—You have called TraceDisableActivity after TraceBegin and before TraceEnd

Usage

Use this function to disable the logging of the specified trace activities. You typically use this function if you are tracing only portions of an application run (and thus you are calling TraceBegin multiple times) and you want to log different activities during each portion of the application.

Unless specifically disabled with TraceDisableActivity, activities that were previously enabled with a call to the TraceEnableActivity function remain enabled throughout the entire application run.

You must always call the TraceEnd function before calling TraceDisableActivity.

Examples

This example logs the enabled activities for the first block of code to be traced. Then it stops logging and disables two activity types for a second trace block. When logging is resumed for another portion of the application run, the activities that are not specifically disabled remain enabled until TraceClose is called:

```
TraceEnableActivity(ActESQL!)
TraceEnableActivity(ActGarbageCollect)
TraceEnableActivity(ActObjectCreate!)
TraceEnableActivity(ActObjectDestroy!)

TraceBegin("Trace_block_1")

TraceEnd()

TraceDisableActivity(ActESQL!)
TraceDisableActivity(ActGarbageCollect!)

TraceBegin("Trace_block_2")
```

See also

TraceEnd
TraceEnableActivity

TraceEnableActivity

Description

Enables logging of the specified trace activity.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

TraceEnableActivity (*activity*)

Argument	Description
activity	A value of the enumerated datatype TraceActivity that identifies the activity to be logged. Values are: • ActError!—Occurrences of system errors and warnings • ActESQL!—Embedded SQL statement entry and exit • ActGarbageCollect!—Start and finish of garbage collection • ActLine!—Routine line hits (if this value is enabled, ActRoutine! is automatically enabled) • ActObjectCreate!—Object creation entry and exit • ActObjectDestroy!—Object destruction entry and exit • ActProfile!—Abbreviation for the ActRoutine!, ActESQL!, ActObjectCreate!, ActObjectDestroy, and ActGarbageCollect! values • ActRoutine!—Routine entry and exit • ActTrace!—Abbreviation for all activities except ActLine!

Return value

ErrorReturn. Returns one of the following values:

- Success!—The function succeeded
- FileNotOpenError!—TraceOpen has not been called yet
- TraceStartedError!—You have called TraceEnableActivity after TraceBegin and before TraceEnd

Usage

Call the TraceEnableActivity function following the TraceOpen function. TraceEnableActivity allows you to specify the types of activities you want logged in the trace file. The default activity type logged is a user-defined activity type identified by the value ActUser!. This activity is enabled by the TraceOpen call. You must call TraceEnableActivity to specify the activities to be logged before you call TraceBegin.

Each call to TraceOpen resets the activity types to be logged to the default (that is, only ActUser! activities are logged).

Since the ActError! and ActUser! values require the passing of strings to the trace file, you must call the TraceError and TraceUser functions to log this information.

Unless specifically disabled with a call to the TraceDisableActivity function, activities that are enabled with TraceEnableActivity remain enabled throughout the entire application run.

Examples

This example opens a trace file with the name you entered in a single line edit box and a timer kind selected from a drop-down list. Then it begins logging the enabled activities for the first block of code to be traced:

```
TimerKindltk_kind

CHOOSE CASE ddlb_timestamp.text
CASE "None"
      ltk_kind = TimerNone!
CASE "Clock"
      ltk_kind = Clock!
CASE "Process"
      ltk_kind = Process!
CASE "Thread"
      ltk_kind = Thread!
END CHOOSE

TraceOpen(sle_filename.text,ltk_kind)

TraceEnableActivity(ActRoutine!)
TraceEnableActivity(ActESQL!)
TraceEnableActivity(ActGarbageCollect!)
TraceEnableActivity(ActError!)
TraceEnableActivity(ActCreateObject!)
TraceEnableActivity(ActDestroyObject!)

TraceBegin("Trace_block_1")
```

See also

TraceOpen
TraceBegin
TraceDisableActivity

TraceEnd

Description

Inserts an activity type value in the trace file indicating that logging has ended and then stops logging application trace activities.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

TraceEnd ( )

Return value

ErrorReturn. Returns one of the following values:

- Success!—The function succeeded
- FileNotOpenError!—TraceOpen has not been called yet
- TraceNotStartedError!—TraceBegin has not been called yet

Usage

The TraceEnd call inserts an activity type value of ActBegin! in the trace file to indicate that logging has ended and then stops logging all application activities that you selected for tracing.

If you have not already called TraceEnd when you call TraceClose, TraceClose calls TraceEnd before proceeding.

If you want to generate a trace file for an entire application run, you would typically include the TraceEnd function in your application's Close script. If you want to generate a trace file for only a portion of the application run, you typically include the TraceEnd function in the script that terminates the functionality on which you're trying to collect data.

Examples

This example stops logging of application trace activities and then closes the open trace file:

```
TraceEnd()
TraceClose()
```

See also

TraceOpen
TraceBegin
TraceClose
TraceDisableActivity

TraceError

Description

Logs your own error message and its severity level to the trace file if tracing of this activity type has been enabled.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

TraceError (*severity*, *message*)

Argument	Description
severity	A long whose value is a number you want to indicate the severity of the error

Argument	Description
message	A string whose value is the error message you want to add to the trace file

Return value

ErrorReturn. This function always returns Success!.

If *severity* or *message* is null, TraceError returns null and no entry is made in the trace file.

Usage

TraceError logs an activity type value of ActError! to the trace file if you enabled the tracing of this type with the TraceEnableActivity function and then called the TraceBegin function. You use the TraceError function to record your own error message. It works just like the TraceUser function except that you use it to identify more severe problems. The *severity* and *message* values are passed without modification to the trace file.

Examples

This example logs an error message to the trace file when a database retrieval fails:

```
dw_1.SetTransObject(SQLCA)

TraceUser(100, "Starting database retrieval")
IF dw_1.Retrieve() = -1 THEN
      TraceError(999, "Retrieve for dw_1 failed")
ELSE
      TraceUser(200, "Database retrieval complete")
END IF
```

See also

TraceEnableActivity
TraceUser

TraceOpen

Description

Opens a trace file with the specified name and enables logging of application trace activities.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

TraceOpen (*filename*, *timer*)

Argument	Description
filename	A read-only string used to identify the trace file
timer	A value of the enumerated datatype TimerKind that identifies the timer. Values are: • Clock!—Use the wall clock timer • Process!—Use the process timer • Thread!—Use the thread timer • TimeNone!—Do not log timer values Clock timers and thread timers are the only kinds of timers supported on handheld devices. If you select the process timer for an application running on a device, the thread timer is used instead.

Return value

ErrorReturn. Returns one of the following values:

- Success!—The function succeeded
- FileAlreadyOpenError!—TraceOpen has been called again without an intervening TraceClose
- FileOpenError!—The file could not be opened for writing
- EnterpriseOnlyFeature!—This function is only supported in the Enterprise edition of PowerBuilder.

If *filename* is null, TraceOpen returns null.

Usage

TraceOpen opens the specified trace file and enables logging of application trace activities. When it opens the trace file, TraceOpen logs the current application and library list to the trace file. It also enables logging of the default activity type, a user-defined activity type identified by the value ActUser!.

After calling TraceOpen, you can select any additional activities to be logged in the trace file using the TraceEnableActivity function. Once you have called TraceOpen and TraceEnableActivity, you must then call TraceBegin for logging to begin.

To stop logging of application trace activity, you must call the TraceEnd function followed by TraceClose to close the trace file. Each call to TraceOpen resets the logging of activity types to the default ActUser!

You typically include the TraceOpen function in your application's Open script.

Caution
If the trace file runs out of disk space, no error is generated, but logging is stopped, and the trace file cannot be used for analysis.

By default, the time at which each activity begins and ends is recorded using the clock timer, which measures an absolute time with reference to an external activity, such as the machine's startup time. The clock timer measures time in microseconds. Depending on the speed of your machine's central processing unit, the clock timer can offer a resolution of less than one microsecond. A timer's resolution is the smallest unit of time the timer can measure.

You can also use process or thread timers in a desktop application. These timers measure time in microseconds with reference to when the process or thread being executed started. You can use only the clock or thread timers for applications running on handheld devices. Both process and thread timers give you a more accurate measurement of how long the process or thread is taking to execute, but both have a lower resolution than the clock timer.

If your analysis does not require timing information, you can omit timing information from the trace file.

Collection time The timestamps in the trace file exclude the time taken to collect the trace data.

Examples

This example opens a trace file with the name you entered in a single line edit box and a timer kind selected from a drop-down list. Then it begins logging the enabled activities for the first block of code to be traced:

```
TimerKindltk_kind

CHOOSE CASE ddlb_timestamp.text
CASE "None"
      ltk_kind = TimerNone!
CASE "Clock"
      ltk_kind = Clock!
CASE "Process"
      ltk_kind = Process!
CASE "Thread"
      ltk_kind = Thread!
END CHOOSE

TraceOpen(sle_filename.text,ltk_kind)
```

See also

TraceBegin
TraceClose
TraceEnableActivity
TraceEnd

TraceUser

Description

Logs the activity type value you specify to the trace file.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

TraceUser (*info*, *message*)

Argument	Description
info	A long whose value is a reference number you want to associate with the logged activity
message	A string whose value is the activity type value you want to add to the trace file

Return value

ErrorReturn. This function always returns Success!.

If *info* or *message* is null, TraceUser returns null and no entry is made in the log file.

Usage

TraceUser logs an activity type value of ActUser! to the trace file. This is the default activity type and is enabled when the TraceOpen function is called. You use the TraceUser function to record your own message identifying a specific occurrence during an application run. For example, you may want to log the occurrences of a specific return value or the beginning and end of a body of code. TraceUser works just like the TraceError function except that you use TraceError to identify more severe problems. The *info* and *message* values are passed without modification to the trace file.

Examples

This example logs user messages to the trace file identifying when a database retrieval is started and when it is completed:

```
dw_1.SetTransObject(SQLCA)

TraceUser(100, "Starting database retrieval")
IF dw_1.Retrieve() = -1 THEN
     TraceError(999, "Retrieve for dw_1 failed")
```

```
ELSE
      TraceUser(200, "Database retrieval complete")
END IF
```

See also

TraceEnableActivity
TraceError

TriggerEvent

Description

Triggers an event associated with the specified object, which executes the script for that event immediately.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Any object

Syntax

objectname.**TriggerEvent** (*event* {, *word*, *long* })

Argument	Description
objectname	The name of any PocketBuilder object or control that has events associated with it.
event	A value of the TrigEvent enumerated datatype that identifies a PocketBuilder event (for example, Clicked!, Modified!, or DoubleClicked!) or a string whose value is the name of an event. The event must be a valid event for *objectname* and a script must exist for the event in *objectname*.
word (optional)	A long value to be stored in the WordParm property of the system's Message object. If you want to specify a value for *long*, but not *word*, enter 0. (For cross-platform compatibility, WordParm and LongParm are both longs.)
long (optional)	A long value or a string that you want to store in the LongParm property of the system's Message object. When you specify a string, a pointer to the string is stored in the LongParm property, which you can access with the String function (see Usage).

Return value

Integer. Returns 1 if it is successful and the event script runs and -1 if the event is not a valid event for *objectname*, or no script exists for the event in *objectname*. If any argument's value is null, TriggerEvent returns null.

Usage

If you specify the name of an event instead of a value of the TrigEvent enumerated datatype, enclose the name in double quotation marks.

Check return code
It is a good idea to check the return code to determine whether TriggerEvent succeeded and, based on the result, perform the appropriate processing.

You can pass information to the event script with the *word* and *long* arguments. The information is stored in the Message object. In your script, you can reference the WordParm and LongParm fields of the Message object to access the information.

If you have specified a string for *long*, you can access it in the triggered event by using the String function with the keyword "address" as the *format* parameter. Your event script might begin as follows:

```
string PassedString
PassedString = String(Message.LongParm, "address")
```

Caution
Do not use this syntax unless you are certain the *long* argument contains a valid string value.

For more information about events and when to use PostEvent and TriggerEvent, see PostEvent.

To trigger system events that are not PocketBuilder-defined events, use Post or Send, instead of PostEvent and TriggerEvent. Although Send can send messages that trigger PocketBuilder events, as shown below, you have to know the codes for a particular message. It is easier to use the PocketBuilder functions that trigger the desired events.

Equivalent syntax Both of the following statements click the CheckBox cb_OK. The following call to the Send function:

```
Send(Handle(Parent), 273, 0, Long(Handle(cb_OK), 0))
```

is equivalent to:

```
cb_OK.TriggerEvent(Clicked!)
```

Examples

This statement executes the script for the Clicked event in the CommandButton cb_OK immediately:

```
cb_OK.TriggerEvent(Clicked!)
```

This statement executes the script for the user-defined event cb_exit_request in the parent window:

```
Parent.TriggerEvent("cb_exit_request")
```

This statement executes the script for the Clicked event in the menu selection m_File on the menu m_Appl:

```
m_Appl.m_File.TriggerEvent(Clicked!)
```

See also

Post
PostEvent
Send

TriggerPBEvent

Description

Triggers the specified user event in the child window contained in a PowerBuilder window ActiveX control.

PocketBuilder	✗
PowerBuilder	✓

Applies to

Window ActiveX controls

Syntax

activexcontrol.**TriggerPBEvent** (*name* {, *numarguments* {, *arguments* } })

Return value

Integer. Returns 1 if the function succeeds and -1 if an error occurs.

Trim

Description

Removes leading and trailing spaces from a string.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

Trim (*string*)

Argument	Description
string	The string you want returned with leading and trailing spaces deleted

Return value

String. Returns a copy of *string* with all leading and trailing spaces deleted if it succeeds and the empty string ("") if an error occurs. If *string* is null, Trim returns null.

Usage

Trim is useful for removing spaces that a user may have typed before or after newly entered data.

Examples

This statement returns BABE RUTH:

```
Trim(" BABE RUTH ")
```

This example removes the leading and trailing spaces from the user-entered value in the SingleLineEdit sle_emp_fname and saves the value in emp_fname:

```
string emp_fname
emp_fname = Trim(sle_emp_fname.Text)
```

See also

LeftTrim
RightTrim
Trim method for DataWindows in the *DataWindow Reference*

TrimW

Description

Removes leading and trailing spaces from a string.

PocketBuilder	✗
PowerBuilder	✓

Obsolete function
TrimW is an obsolete function. It has the same behavior as Trim.

Syntax

TrimW (*string*)

Return value

String. Returns a copy of *string* with all leading and trailing spaces deleted if it succeeds and the empty string ("") if an error occurs.

Truncate

Description

Truncates a number to the specified number of decimal places.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

Truncate (*x*, *n*)

Argument	Description
x	The number you want to truncate
n	The number of decimal places to which you want to truncate *x* (valid values are 0 through 18)

Return value

Decimal. Returns the result of the truncation if it succeeds and null if it fails or if any argument is null.

Using Truncate on a computed field
A real number loaded into a floating point register (used for calculation) is represented as precisely as the binary storage will permit. For example, the real number displayed as 2.07 is actually stored as 2.0699999999999997.

Truncating such a number may not give the expected result. To avoid this problem, you can change the initial real datatype to long, Integer, or decimal, or you can append a constant in the truncate argument:
Truncate (*x* + 0.0000001, *n*)

Examples

This statement returns 9.2:

```
Truncate(9.22, 1)
```

This statement returns 9.2:

```
Truncate(9.28, 1)
```

This statement returns 9:

```
Truncate(9.9, 0)
```

This statement returns –9.2:

```
Truncate(-9.29, 1)
```

See also

Ceiling
Int
Round
Truncate method for DataWindows in the *DataWindow Reference*

TrustVerify

Description

Called by EAServer when an SSL certificate chain needs to be approved for use by a client. This function is used by PowerBuilder clients connecting to EAServer.

PocketBuilder	✗
PowerBuilder	✓

Applies to

SSLCallBack objects

Syntax

sslcallback.**TrustVerify** (*thesessioninfo, reason*)

Return value

Long. Returns one of the following values:

1	TRUST_ONCE (accept the current connection)
2	TRUST_FAIL (reject the current connection)
3	TRUST_ALWAYS (accept and mark as trusted in the database)
4	TRUST_NEVER (reject and mark as untrusted in the database)
5	TRUST_SESSION (accept now and throughout the current session)
6	TRUST_FAIL_SESSION (reject throughout the current session)

TypeOf

Description

Determines the type of an object or control, reported as a value of the Object enumerated datatype.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Any object

Syntax

objectname.**TypeOf** ( )

Argument	Description
objectname	The name of the object or control for which you want the type

Return value

Object enumerated datatype. Returns the type of *objectname*. If *objectname* is null, TypeOf returns null.

Usage

Use TypeOf to determine the type of a selected or dragged control.

Examples

If dw_Customer is a DataWindow control, this statement returns DataWindow!:

```
dw_Customer.Typeof()
```

This example looks at the first five controls in the w_dept window's Control array property. The loop executes some statements for each control that is a CheckBox:

```
integer n
FOR n = 1 to 5
      IF w_dept.Control[n].TypeOf() = CheckBox! THEN
      ... // Some processing
      END IF
NEXT
```

This loop stores in the winobject array the type of each object in the window's Control array property:

```
object winobjecttype[]
long ll_count

FOR ll_count = 1 to UpperBound(Control[])
      winobjecttype[ll_count] = &
         TypeOf(Control[ll_count])
NEXT
```

If you do not know the type of a control passed via PowerObjectParm in the Message object, the following example assigns the passed object to a graphic object variable, the ancestor of all the control types, and assigns the type to a variable of type object, which is the enumerated datatype that TypeOf returns. The CHOOSE CASE statement can include processing for each control type that you want to handle. This code would be in the Open event for a window that was opened with OpenWithParm:

```
graphicobject stp_obj
object type_obj

stp_obj = Message.PowerObjectParm
type_obj = stp_obj.TypeOf()
```

```
CHOOSE CASE type_obj
CASE DataWindow!
      MessageBox("The object"," Is a datawindow")

CASE SingleLineEdit!
      MessageBox("The object"," Is a sle")

... // Cases for additional object types
CASE ELSE
      MessageBox("The object"," Is irrelevant!")
END CHOOSE
```

See also

ClassName

Uncheck

Description

Removes the check mark, if any, next to an item a drop-down or cascading menu and sets the item's Checked property to false.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Menu objects

Syntax

menuname.**Uncheck** ( )

Argument	Description
menuname	The fully qualified name of the menu selection from which you want to remove the checkmark, if any. The menu must be on a drop-down or cascading menu, not an item on a menu bar.

Return value

Integer. Returns 1 if it succeeds and -1 if an error occurs. If *menuname* is null, Uncheck returns null.

Usage

A checkmark next to a menu item indicates that the menu option is currently on and that the user can turn the option on and off by choosing it. For example, in the Window painter's Design menu, a checkmark is displayed next to Grid when the grid is on.

You can use Check in an item's Clicked script to mark a menu item when the user turns the option on and Uncheck to remove the check when the user turns the option off.

Equivalent syntax You can set the object's Checked property instead of calling Uncheck:

menuname.Checked = false

This statement:

```
m_appl.m_view.m_grid.Checked = FALSE
```

is equivalent to:

```
m_appl.m_view.m_grid.Uncheck()
```

Examples

This statement removes the checkmark next to the m_grid menu selection in the drop-down menu m_view on the menu bar m_appl:

```
m_appl.m_view.m_grid.Uncheck()
```

This example checks whether the m_grid menu selection in the drop-down menu m_view of the menu bar m_appl is currently checked. If so, the script unchecks the item. If it is not checked, the script checks the item:

```
IF m_appl.m_view.m_grid.Checked = TRUE THEN
      m_appl.m_view.m_grid.Uncheck()
ELSE
      m_appl.m_view.m_grid.Check()
END IF
```

See also

Check

Undo

Description

Cancels the last edit in an edit control, restoring the text to the content before the last change.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

DataWindow, MultiLineEdit, RichTextEdit, and SingleLineEdit controls

Syntax

editname.**Undo** ( )

Argument	Description
editname	The name of the control in which you want to cancel (reverse) the last edit. For a DataWindow control, reverses the last edit in the edit control over the current row and column.

Return value

Integer. Returns 1 when it succeeds and -1 if an error occurs. If *editname* is null, Undo returns null.

Usage

To determine whether the last action can be canceled, call the CanUndo function.

Examples

This statement reverses the last edit in MultiLineEdit mle_Contact:

```
mle_Contact.Undo()
```

The following statement checks to see if the last edit in the MultiLineEdit mle_Contact can be reversed, and if so reverse it:

```
IF mle_Contact.CanUndo() THEN mle_Contact.Undo()
```

See also

CanUndo

UnitsToPixels

Description

Converts PowerBuilder units to pixels and reports the measurement. Because pixels are not usually square, you also specify whether to convert in the horizontal or vertical direction.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

UnitsToPixels (*units*, *type*)

Argument	Description
units	An integer whose value is the number of PowerBuilder units you want to convert to pixels
type	A value of the ConvertType enumerated datatype indicating how to convert the value: • XUnitsToPixels! — Convert the units in the horizontal direction • YUnitsToPixels! — Convert the units in the vertical direction

Return value

Integer. Returns the converted value if it succeeds and -1 if an error occurs. If any argument's value is null, UnitsToPixels returns null.

Examples

These statements convert 350 vertical PowerBuilder units to vertical pixels and set value equal to the converted value:

```
integer Value
Value = UnitsToPixels(350, YUnitsToPixels!)
```

See also

PixelsToUnits

Update

Updates a change to an object or to a repository item at runtime.

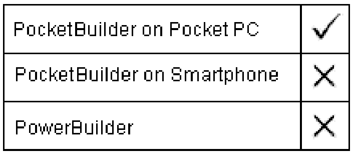
For syntax for DataWindows and DataStores, see the Update method for DataWindows in the online Help.

To update	Use
A NotificationBubble object	Syntax 1
An appointment, contact, or task from Pocket Outlook	Syntax 2

Syntax 1 For NotificationBubble objects

Description

Notifies the Windows CE operating system that properties of a notification bubble control have changed.



PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✗

Applies to

NotificationBubble objects

Syntax

Integer *controlname*.Update ( )

Argument	Description
controlname	The name of the notification bubble that has been created

Return value

Integer. Returns 1 for success and one of the following negative values if an error occurs:

-1 Initial creation notification failed

-2 Notification failed

-3 Notification received, but nothing changed

-4 Mandatory message sink has not been specified

Usage

The Update function is the main action method for a NotificationBubble object. The first time it is called, it creates the notification bubble in the operating system. Subsequent calls notify the operating system that notification fields have changed.

The NotificationBubble must be associated with a visual control. You assign the visual control with the SetMessageSink function. If the NotificationBubble object is not associated with a visual control, the Update function returns a -4 error.

Examples

The following example notifies Windows CE that a notification event has occurred and that the nb_myBubble NotificationBubble object has been created or updated:

```
nb_myBubble.caption = "Updated Caption"
li_rtn = nb_myBubble.Update()
```

See also

Remove
SetMessageSink

Syntax 2 For POOM-related objects

Description

Updates an existing appointment, contact, or task in the POOM repository.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✗

Applies to

POOMAppointment, POOMContact, POOMTask objects

Syntax

Integer *objectname*.Update ( )

Argument	Description
objectname	The name of the POOMAppointment, POOMContact, or POOMTask object

Return value

Integer. Returns 1 for success and one of the following negative values if an error occurs:

-1 Unspecified error

-2	Cannot connect to the repository or a required internal subobject failed to connect to the repository
-3	Cannot log in to the repository
-4	Incorrect input argument
-5	Action cannot be performed
-6	The object identifier (OID) is not in the repository
-7	Feature is not implemented yet
-8	No matching entries found for the criteria

See also

Add
Cancel
Remove

UpdateEntry

Description

Updates an entry in a dialing directory.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✗

Applies to

DialingDirectory objects

Syntax

Integer *objectname*.UpdateEntry (*entry*)

Argument	Description
objectname	The name of the DialingDirectory object to which you want to add an entry.
entry	A DialingDirectoryEntry structure that holds the replacement value.

Return value

Integer. Returns 1 for success, and a negative value if an error occurs.

See also

AcceptCall
GetEntry
GetEntries
Update for POOM-related objects

UpdateLinksDialog

Description

Attempts to find a file linked to an OLE container. If the linked file is not found, a dialog box tells the user and lets them bring up a second dialog box for find the file or changing the link.

PocketBuilder	✗
PowerBuilder	✓

Applies to

OLE controls and OLE DWObjects (objects within a DataWindow object that is within a DataWindow control)

Syntax

objectref.**UpdateLinksDialog** ( )

Return value

Integer. Returns 0 if it succeeds and -1 if an error occurs.

Upper

Description

Converts all the characters in a string to uppercase.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

Upper (*string*)

Argument	Description
string	The string you want to convert to uppercase letters

Return value

String. Returns *string* with lowercase letters changed to uppercase if it succeeds and the empty string ("") if an error occurs. If *string* is null, Upper returns null.

Examples

This statement returns `BABE RUTH`:

```
Upper("Babe Ruth")
```

See also

Lower
Upper method for DataWindows in the *DataWindow Reference*

UpperBound

Description

Obtains the upper bound of a dimension of an array.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

UpperBound (*array* {, *n* })

Argument	Description
array	The name of the array for which you want the upper bound of a dimension
n (optional)	The number of the dimension for which you want the upper bound. The default is 1

Return value

Long. Returns the upper bound of dimension *n* of *array*. If *n* is greater than the number of dimensions of the array, UpperBound returns -1. If any argument's value is null, UpperBound returns null.

Usage

For variable-size arrays, memory is allocated for the array when you assign values to it. UpperBound returns the largest value that has been defined for the array in the current script. Before you assign values, the lower bound is 1 and the upper bound is 0. For fixed arrays, whose size is specified when it is declared, UpperBound always returns the declared size.

Examples

The following statements illustrate the values UpperBound reports for fixed-size arrays and for variable-size arrays before and after memory has been allocated:

```
integer a[5]
UpperBound(a)    // Returns 5
UpperBound(a,1)  // Returns 5
UpperBound(a,2)  // Returns -1; no 2nd dimension

integer b[10,20]
UpperBound(b,1)  // Returns 10
UpperBound(b,2)  // Returns 20

integer c[ ]
UpperBound(c)   // Returns 0; no memory allocated
c[50] = 900
UpperBound(c)   // Returns 50
c[60] = 800
UpperBound(c)   // Returns 60
```

```
c[60] = 800
c[50] = 700
UpperBound(c)   // Returns 60

integer d[10 to 50]
UpperBound(d)   // Returns 50
```

This example determines the position of a menu bar item called File, and if the item has a cascading menu with an item called Update, disables the Update item. The code could be a script for a control in a window.

The code includes a rather complicated construct: Parent.Menuid.Item. Its components are:

- Parent — The parent window of the control that is running the script.
- Menuid — A property of a window whose value identifies the menu associated with the window.
- Item — A property of a menu that is an array of items in that menu. If Item is itself a drop-down or cascading menu, it has its own item array, which can be a fourth qualifier.

The script is:

```
long i, k, tot1, tot2

// Determine how many menu bar items there are.
tot1 = UpperBound(Parent.Menuid.Item)

FOR i = 1 to tot1
      // Find the position of the File item.
      IF Parent.Menuid.Item[i].text = "File" THEN
         MessageBox("Position", &
            "File is in Position "+ string(i))
         tot2 = UpperBound(Parent.Menuid.Item[i].Item)

         FOR k = 1 to tot2
            // Find the Update item under File.
            IF Parent.Menuid.Item[i].Item[k].Text = &
               "Update" THEN
               // Disable the Update menu option.
               Parent.Menuid.Item[i].Item[k].Disable()
               EXIT
            END IF
         NEXT
         EXIT
      END IF
```

```
NEXT
```

See also LowerBound

VerifyMatch

Description Verifies the similarity between two fingerprints.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✗

Applies to BiometricScanner objects

Syntax Integer *scanner*.VerifyMatch (*candidate*, *template*, {*FARAchieved*})

Argument	Description
scanner	The scanner object associated with the device you want to use to complete a scan
candidate	Blob value for the current minutiae data that you want to compare
template	Blob value for a stored minutiae record
FARAchieved	Integer value, passed by reference, for the false acceptance rate of the most recent scan

Return value Integer. Returns 1 for a successful match within the specified FAR/FRR ratio. A return value of -14 indicates that the comparison value falls outside this ratio. For a list of all possible errors and their definitions, see ScanCapture on page 868.

Usage Call VerifyMatch to compare two fingerprint scans. Typically the result of a current candidate scan is compared against a fingerprint scan stored in a database. The scan stored in the database is also known as a template scan.

Examples The following example compares the scanned data against a local variable with a blob datatype:

```
li_rtn = l_bioscanner.VerifyMatch (lb_MinutiaeFromScan,
lb_MinutiaeFromTemplate)
```

See also ScanCapture
ScannedBitmap
ScannedMinutiae
ScannedQuality

Which

Description

Allows a component to find out whether it is running on a transaction server.

PocketBuilder	✗
PowerBuilder	✓

Applies to

TransactionServer objects

Syntax

transactionserver.**Which** ( )

Return value

Integer. Returns 0 if the object is not running on a transaction server, 1 if it is running on EAServer, or 2 if it is running on Microsoft MTS or IIS4.

WordCap

Description

Capitalizes the first letter of each word in a passed script. It sets the remaining letters in each word to lowercase.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

All text objects

Syntax

WordCap (*text*)

Argument	Description
text	String to be modified

Return value

String. If it succeeds, returns the text passed in the function argument with the first letter of each word in uppercase and the remaining letters in lowercase. Returns null if an error occurs.

Examples

This example takes user-entered text from a SingleLineEdit control, capitalizing the first letter in each word and setting the other letters to lowercase, before passing it in a string variable:

```
string ls_fullname
ls_fullname = WordCap (sle_1.text)
```

The text `joe MaCdonald` would be rendered as `Joe Macdonald` by the WordCap function.

WorkSpaceHeight

Description

Obtains the height of the workspace within the boundaries of the specified window.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Window objects

Syntax

windowname.**WorkSpaceHeight** ( )

Argument	Description
windowname	The name of the window for which you want the height of the workspace area

Return value

Integer. Returns the height of the workspace area in PowerBuilder units in *windowname*. If an error occurs, WorkSpaceHeight returns -1. If *windowname* is null, WorkSpaceHeight returns null.

Usage

The workspace height does not include the thickness of the frame, the title bar, menu bar, horizontal scrollbar, or any toolbars at the top or bottom.

The workspace width does not include the thickness of the frame, the vertical scrollbar, or any toolbars on the left or right.

Examples

This example returns the height of the workspace area in the w_employee window:

```
Integer Height
Height = W_employee.WorkSpaceHeight()
```

See also

WorkSpaceWidth
WorkSpaceX
WorkSpaceY
PointerX
PointerY

WorkSpaceWidth

Description

Obtains the width of the workspace within the boundaries of the specified window.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Window objects

Syntax

windowname.**WorkSpaceWidth** ( )

Argument	Description
windowname	The name of the window for which you want the width of the workspace area

Return value

Integer. Returns the width of the workspace area (in PowerBuilder units) in *windowname*. If an error occurs, WorkSpaceWidth returns -1. If *windowname* is null, WorkSpaceWidth returns null.

Usage

The workspace height does not include the thickness of the frame, the title bar, menu bar, horizontal scrollbar, or any toolbars at the top or bottom.

The workspace width does not include the thickness of the frame, the vertical scrollbar, or any toolbars on the left or right.

Examples

This example returns the width of the workspace area in the w_employee window:

```
integer Width
Width = w_employee.WorkSpaceWidth()
```

See also

PointerX
PointerY
WorkSpaceHeight
WorkSpaceX
WorkSpaceY

WorkSpaceX

Description	Obtains the distance between the left edge of a window's workspace and the left edge of the screen.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to	Window objects
Syntax	*windowname*.**WorkSpaceX** ( )

Argument	**Description**
windowname	The name of the window for which you want the distance between the left edge of the workspace area and the left edge of the screen

Return value

Integer. Returns the distance that the left edge of the workspace area of *windowname* is from the left edge of the screen (in PowerBuilder units). WorkSpaceX returns -1 if an error occurs. If *windowname* is null, WorkSpaceX returns null.

Usage

The workspace area is the area between the sides of the window (not including the thickness of the frame or the vertical scrollbar, if any) and the top and bottom of the window (not including the thickness of the frame or the title bar, menu bar, or horizontal scrollbar, if any).

Examples

This example returns the distance from the left edge of the screen to the left edge of the workspace area in the w_employee window:

```
integer workx
workx = w_employee.WorkSpaceX()
```

See also

PointerX
PointerY
WorkSpaceHeight
WorkSpaceWidth
WorkSpaceY

WorkSpaceY

Description

Obtains the distance between the top of a window's workspace and the top of the screen.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Window objects

Syntax

windowname.**WorkSpaceY** ( )

Argument	Description
windowname	The name of the window for which you want the distance between the top of the workspace area and the top of the screen

Return value

Integer. Returns the distance that the top of the workspace area of *windowname* is from the top of the screen (in PowerBuilder units). If an error occurs, WorkSpaceY returns -1. If *windowname* is null, WorkSpaceY returns null.

Usage

The workspace area is the area between the sides of the window (not including the thickness of the frame or the vertical scrollbar, if any) and the top and bottom of the window (not including the thickness of the frame or the title bar, menu bar, or horizontal scrollbar, if any).

Examples

This example returns the distance from the top of the screen to the top of the workspace area in the w_employee window:

```
integer worky
worky = w_employee.WorkSpaceY()
```

See also

PointerX
PointerY
WorkSpaceHeight
WorkSpaceWidth
WorkSpaceX

Write

Writes data to an OLE stream object or a file that you open with the FileDirect object.

To	Use
Write data to an OLE stream object	Syntax 1
Write data into an array	Syntax 2
Write data into a blob	Syntax 3

Syntax 1 For an OLE stream object

Description

Writes data to an opened OLE stream object.

PocketBuilder	✗
PowerBuilder	✓

Applies to

OLEStream objects

Syntax

olestream.**Write** (*dataforstream*)

Return value

Integer. Returns the number of characters or bytes written if it succeeds and A negative integer if an error occurs.

Syntax 2 For writing data from an array of bytes

Description

Writes data from an array into an open file.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✗

Applies to

FileDirect objects

Syntax

Integer *instancename*.Write (*data[ ]*, *bytecount*)

Argument	Description
instancename	Name of the instance of the FileDirect object
data[ ]	An array of integers representing bytes of data
bytecount	Integer for the number of bytes that you want to write in the open file

Return value

Integer. Returns 1 for success and a negative number for any error.

Usage

Use this function to write to a file that you open with the FileDirect object in write mode. The FileDirect object supports only the synchronous style of file input; further file-related commands cannot be called until after the Write function is fully processed or an error in writing to the file is caught.

Examples

The following example calls the FileDirect user object nvo_fileDirect to open a file, write some data, and close the file:

```
Integer li_ret, li_AmountRead, li_data [ ]
li_ret = nvo_fileDirect.Open ("COM8:", stgReadWrite!)
li_ret = nvo_fileDirect.Write (li_data[], 100)
li_ret = nvo_fileDirect.Close ( )
```

See also

Read

Syntax 3 For writing data from a blob

Description

Writes data from a blob to an open file.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✗

Applies to

FileDirect objects

Syntax

Integer *instancename*.Write (*bdata*, *bytecount*)

Argument	Description
instancename	Name of the instance of the FileDirect object
bdata	A blob variable holding the data that you write to a file
bytescount	Integer for the number of bytes that you want to write to the open file

Return value

Integer. Returns 1 for success and a negative number for any error.

Usage

Use this function to write to a file that you open with the FileDirect object in write mode. The FileDirect object supports only the synchronous style of file input; further file-related commands cannot be called until after the Write function is successfully processed or until an error in writing to the file is caught.

Examples

The following example calls the FileDirect user object nvo_fileDirect to open a file, write some data, and close the file:

```
Integer li_ret, li_AmountRead
Blob lb_data
li_ret = nvo_fileDirect.Open ("MyDoc.txt", stgRead!)
li_ret = nvo_fileDirect.Write (lb_data, 100)
li_ret = nvo_fileDirect.Close ( )
```

See also

Open
Read

Year

Description

Determines the year of a date value.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

Year (*date*)

Argument	Description
date	The date from which you want the year

Return value

Integer. Returns an integer whose value is a 4-digit year adapted from the year portion of *date* if it succeeds and 1900 if an error occurs. If *date* is null, Year returns null.

When you convert a string that has a two-digit year to a date, then PocketBuilder chooses the century, as follows. If the year is between 00 to 49, PocketBuilder assumes 20 as the first two digits; if it is between 50 and 99, PocketBuilder assumes 19.

Usage

PocketBuilder handles years from 1000 to 3000 inclusive.

If your data includes date before 1950, such as birth dates, always specify a 4-digit year so that Year and other PocketBuilder functions, such as Sort, interpret the date as intended.

Windows settings
To make sure you get correct return values for the year, you must verify that yyyy is the Short Date Style for year in the Regional Settings of the user's Control Panel. Your program can check this with the RegistryGet function.

If the setting is not correct, you can ask the user to change it manually or have the application change it (by calling the RegistrySet function). The user may need to reboot after the setting is changed.

Examples

This statement returns 1995:

```
Year(1995-01-31)
```

See also

Day
Month
Year method for DataWindows in the *DataWindow Reference*

Yield

Description

Yields control to other graphic objects, including objects that are not PocketBuilder objects. Yield checks the message queue and if there are messages in the queue, it pulls them from the queue.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

Yield ( )

Return value

Boolean. Returns true if it pulls messages from the message queue and false if there are no messages.

Usage

Include Yield within a loop so that other processes can happen. For example, use Yield to allow end users to interrupt a loop. By yielding control, you allow the user time to click on a cancel button in another window. Then code in the loop can check whether a global variable's status has changed. You can also use Yield in a loop in which you are waiting for something to finish so that other processing can take place, in either your or some other application.

Using other applications while retrieving data
Although the user cannot do other activities in a PocketBuilder application while retrieving data, you can allow them to use other applications on their system. Put Yield in the RetrieveRow event so that other applications can run during the retrieval.

Of course, Yield will make your PocketBuilder application run slower because processing time will be shared with other applications.

Examples

In this example, some code is processing a long task. A second window includes a button that the user can click to interrupt the loop by setting a shared boolean variable sb_interrupt. When the user clicks the button, its Clicked script sets *sb_interrupt*, shown here:

```
sb_interrupt = TRUE
```

The script that is doing the processing checks the shared variable *sb_interrupt* and interrupts the processing if it is true. The Yield function allows a break in the processing so the user has the opportunity to click the button:

```
integer n
// sb_interrupt is a shared variable.
sb_interrupt = FALSE

FOR n = 1 to 3000
      Yield()
      IF sb_interrupt THEN // var set in other script
         MessageBox("Debug","Interrupted!")
         sb_interrupt = FALSE
         EXIT
      ELSE
      ... // Some processing
      END IF
NEXT
```

In this example, this script doing some processing runs in one window while users interact with controls in a second window. Without Yield, users could click in the second window, but they would not see focus change or their actions processed until the loop completed:

```
integer n

FOR n = 1 to 3000
      Yield()
      ... // Some processing
NEXT
```

In this example, a script wants to open a DDE channel with Lotus Notes, whose executable name is stored in the variable mailprogram. If the program is not running, the script starts it and loops, waiting until the program's startup is finished and it can establish a DDE channel. The loop includes Yield, so that the computer can spend time actually starting the other program:

```
time starttime
long hndl

SetPointer(HourGlass!)
//Try to establish a handle; SendMail is the topic.
hndl = OpenChannel("Notes","SendMail")

//If the program is not running, start it
IF hndl < 1 then
      Run(mailprogram, Minimized!)
      starttime = Now()

      // Wait up to 2 minutes for Notes to load
      // and the user to log on.
      DO
         //Yield control occasionally.
         Yield()
         //Is Notes active yet?
         hndl = OpenChannel("Notes","SendMail")
         // If Notes is active.
         IF hndl > 0 THEN EXIT
      LOOP Until SecondsAfter(StartTime,Now()) > 120

      // If 2 minutes pass without opening a channel
      IF hndl < 1 THEN
         MessageBox("Error", &
            "Can't start Notes.", StopSign!)
         SetPointer(Arrow!)
         RETURN
      END IF
END IF
```

Index