SYBASE®

DataWindow® Reference

PocketBuilder™

2.5

DOCUMENT ID: DC00131-01-0250-01

LAST REVISED: December 2007

Sybase, Inc., One Sybase Drive, Dublin, CA 94568.

Contents

About This Book

Audience

This guide is for anyone defining DataWindow® objects and writing scripts that deal with DataWindow objects. It assumes that:

- You are familiar with the DataWindow painter. If not, see the PocketBuilder™ *Users Guide* or the PowerBuilder® *Users Guide*.
- You have a basic familiarity with PowerScript®. If not, see the *PowerScript Reference*.

How to use this book

This book provides reference information for the DataWindow object. It lists the DataWindow functions and properties and includes the syntax for accessing properties and data.

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.
- *Objects and Controls* - Describes the system-defined objects and their default properties, functions, and events.
- *PowerScript Reference* - Describes syntax and usage for the PowerScript language including variables, expressions, statements, events, and functions.

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 PocketBuilder.

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.

CHAPTER 1

DataWindow Operators and Expressions

About this chapter

You use an expression to request that a DataWindow object or report perform a computational operation. This chapter explains how expressions work and how to write them.

Contents

Where you use DataWindow expressions

A DataWindow expression is a combination of data, operators, and functions that, when evaluated, results in a value. An expression can include column names, operators, DataWindow expression functions, and constants such as numbers and text strings.

In painters

DataWindow expressions are associated with DataWindow objects and reports. You specify them in the DataWindow painter. You can also specify expressions in the Database painter, although these expressions have a slightly different format.

For information about DataWindow expression functions that you can use in expressions, see “Using DataWindow expression functions” on page 15, or look up the function you want in online Help.

In a DataWindow object or report, you use expressions in these ways:

Table 1-1: Using DataWindow expressions in PocketBuilder painters

In this painter	Expressions are used in
DataWindow painter	Computed fields Conditional expressions for property values Validation rules Filters Sorting Series and values in graphs Columns, rows, and values in crosstabs
Database painter	Validation rules

Other types of expressions you use

You also use expressions in Quick Select, SQL Select, and the Query painter to specify selection criteria, and in SQL Select and the Query painter to create computed columns. In these painters you are using SQL operators and DBMS-specific functions, not DataWindow expression operators and functions, to create expressions.

You can access and change the value of DataWindow data and properties in code. The format for expressions you specify in code is different from the same expression specified in the painter. These differences are described in Chapter 4, “Accessing Data in Code” and Chapter 5, “Accessing DataWindow Object Properties in Code”.

Some of the specific places where you use expressions are described here.

In computed fields

Expressions for computed fields can evaluate to any value. The datatype of the expression becomes the datatype of the computed field:

Table 1-2: Using expressions in computed fields

Expression	Description
Today ( )	Displays the date using the Today function.
Salary/12	Computes the monthly salary.
Sum (Salary for group 1)	Computes the salary for the first group using the Sum aggregate function.
Price*Quantity	Computes the total cost.

Expressions for graphs and crosstabs

You can use similar expressions for series and values in graphs and for columns, rows, and values in crosstabs.

In filters

Filter expressions are boolean expressions that must evaluate to TRUE or FALSE:

Table 1-3: Using expressions with filters

Expression	Description
Academics = "*****" AND Cost = "$$$"	Displays data only for colleges with both a 5-star academic rating and a $$$ cost rating.
Emp_sal < 50000	Displays data for employees with salaries less than $50,000.
Salary > 50000 AND Dept_id BETWEEN 400 AND 700	Displays data for employees in departments 400, 500, 600, and 700 with salaries greater than $50,000.
Month(Bdate) = 9 OR Month(Bdate) = 2	Displays data for people with birth dates in September or February.
Match (Lname, "[^ABC]")	Displays data for people whose last name begins with A, B, or C.

In validation rules for table columns

Validation rules are boolean expressions that compare column data with values and that use relational and logical operators. When the validation rule evaluates to FALSE, then the data in the column is rejected.

In the DataWindow painter When you specify a validation rule in the DataWindow painter, you want to validate the newly entered value. To refer to the newly entered value, use the GetText function. Because GetText returns a string, you also need a data conversion function (such as Integer or Real) if you compare the value to other types of data.

If you include the column name in the expression, you get the value that already exists for the column instead of the newly entered value that needs validating.

In the Database painter When you specify the validation rule in the Database painter, you are defining a general rule that can be applied to any column. Use @placeholder to stand for the newly entered value. The name you use for @placeholder is irrelevant—you can assign the rule to any column that has a datatype appropriate for the comparison.

When you define a DataWindow object, a validation rule assigned to a column is brought into the DataWindow object and converted to DataWindow object syntax. @placeholder is converted to GetText and the appropriate datatype conversion function.

Other columns in the rule You can refer to values in other columns for the current row by specifying their names in the validation rule:

Table 1-4: Using expressions with values from other columns

Expression in Database painter	Expression in DataWindow painter	Description
@column >= 10000	Integer(GetText())>= 10000	If a user enters a salary below $10,000, an error message displays.
@column IN (100, 200, 300)	Integer(GetText()) IN (100, 200, 300)	If a user does not enter a department ID of 100, 200, or 300, an error message displays.
@salary > 0	Long(GetText()) > 0	If a user does not enter a positive number, an error message displays.
Match(@disc_price, "^[0-9]+$") and @disc_price < Full_Price	Match(GetText( ), "^[0-9]+$") and Real(GetText()) < Full_Price	If a user enters any characters other than digits, or the resulting number is greater than or equal to the value in the Full_Price column, an error message displays.

Operators used in DataWindow expressions

An operator is a symbol or word in an expression that performs an arithmetic calculation or logical operation; compares numbers, text, or values; or manipulates text strings.

Four types of operators are available:

- **Arithmetic** for numeric datatypes. See “Arithmetic operators in DataWindow expressions” on page 5.
- **Relational** for all datatypes. See “Relational operators in DataWindow expressions” on page 5.
- **Logical** for all datatypes. See “Logical operators in DataWindow expressions” on page 9.
- **Concatenation** for string datatypes. See “Concatenation operator in DataWindow expressions” on page 10.

Arithmetic operators in DataWindow expressions

When you write an expression, you can use the following arithmetic operators:

Table 1-5: Using expressions with arithmetic operators

Operator	Meaning	Example
+	Addition	SubTotal + Tax
-	Subtraction	Price - Discount
*	Multiplication	Quantity * Price
/	Division	Discount / Price
^	Exponentiation	Rating ^ 2.5

Multiplication and division

Multiplication and division are carried out to full precision (16–18 digits). Values are rounded:

Table 1-6: Value rounding in DataWindow expressions

Expression	Value
20.0/3	6.666666666666667
3*(20.0/3)	20
Truncate(20.0/3,4)	6.6666

Calculations with NULL

When you form an arithmetic expression that contains a NULL value, the expression becomes NULL. Thinking of NULL as *undefined* makes this easier to understand. For example, when a NULL column is multiplied by 5, the entire expression also evaluates to NULL. Use the IsNull function to explicitly check for the NULL value.

Boolean expressions that contain a NULL value evaluate to FALSE rather than to NULL. For more information, see “Relational operators in DataWindow expressions” next.

Relational operators in DataWindow expressions

You use relational operators to compare a value with other values. The result is a boolean expression whose value is always TRUE or FALSE.

Since the result of a boolean expression is always TRUE or FALSE, a relational operator that compares a value to NULL, evaluates to FALSE. For example, the expression "column > 5" evaluates to FALSE (and "NOT column > 5" evaluates to TRUE) when the column value is NULL.

When you write an expression, you can use the following relational operators (more information about LIKE, IN, and BETWEEN follows the table):

Table 1-7: Using expressions with relational operators

Operator	Meaning	Example
=	Is equal to	Price = 100
>	Is greater than	Price > 100
<	Is less than	Price < 100
<>	Is not equal to	Price <> 100
>=	Greater than or equal to	Price >= 100
<=	Less than or equal to	Price <= 100
NOT =	Is not equal to	Price NOT = 100
LIKE	Matches this specified pattern.	Emp_lname LIKE 'C%' OR Emp_lname LIKE 'G%'
IN	Is in this set of values.	Dept_id IN (100, 200, 500)
BETWEEN	Is within this range of values. The range includes the first and last values.	Price BETWEEN 1000 AND 3000
NOT LIKE	Does not match this specified pattern.	Emp_lname NOT LIKE 'C%' AND Emp_lname NOT LIKE 'G%'
NOT IN	Is not in this set of values.	Dept_id NOT IN (100, 200, 500)
NOT BETWEEN	Is outside this range of values. The range includes the first and last values.	Price NOT BETWEEN 1000 AND 2000

Special characters for operations with strings

You can use the following special characters with relational operators that take string values:

Table 1-8: Special characters for use in expressions with relational operators

Special character	Meaning	Example
% (percent)	Matches any group of characters.	Good% matches all names that begin with Good.
_ (underscore)	Matches any single character.	Good _ _ _ matches all 7-letter names that begin with Good.

LIKE and NOT LIKE operators

Use LIKE to search for strings that match a predetermined pattern. Use NOT LIKE to search for strings that do not match a predetermined pattern. When you use LIKE or NOT LIKE, you can use the % or _ characters to match unknown characters in a pattern.

For example, the following expression for the Background.Color property of the Salary column displays salaries in red for employees with last names beginning with F and displays all other salaries in white:

```
If(emp_lname LIKE'F%',RGB(255,0,0),RGB(255,255,255))
```

BETWEEN and NOT BETWEEN operators

Use BETWEEN to check if a value is within a range of values. Use NOT BETWEEN to check if a value is *not* in a range of values. The range of values includes the boundary values that specify the range.

For example, the following expression for the Background.Color property of the Salary column displays salaries in red when an employee's salary is between $50,000 and $100,000 and displays all other salaries in white:

```
If(salary BETWEEN 50000 AND 100000, RGB(255,0,0),
RGB(255,255,255))
```

You can use the BETWEEN and NOT BETWEEN operators with string values. For example, if the following expression is used for the Visual property of a column, column values display only for departments listed alphabetically between Finance and Sales:

```
If(dept_name BETWEEN 'Finance' AND 'Sales',1,0)
```

The % or _ characters can be used when you are using string values with the BETWEEN and NOT BETWEEN operators. This example might include more department listings than the previous example:

```
If(dept_name BETWEEN 'F%' AND 'S%',1,0)
```

You can also use the BETWEEN and NOT BETWEEN operators with methods. For example:

```
GetRow( ) BETWEEN 5 AND 8
```

IN and NOT IN operators

Use IN to check if a value is in a set of values. Use NOT IN to check if a value is *not* in a set of values.

For example, the following expression for the Background.Color property of the Salary column displays salaries in red for employees in department 300 or 400 having a salary between $50,000 and $100,000, and displays all other salaries in white:

```
If(dept_id IN (300,400) and salary BETWEEN 50000 AND
100000, RGB(255,0,0), RGB(255,255,255))
```

Comparing strings in DataWindow expressions

When you compare strings, the comparison is case sensitive. Leading blanks are significant, but trailing blanks are not.

Case-sensitivity examples

Assume City1 is "Austin" and City2 is "AUSTIN". Then:

```
City1=City2
```

returns FALSE.

To compare strings regardless of case, use the Upper or Lower function. For example:

```
Upper(City1)=Upper(City2)
```

returns TRUE.

For information about these functions, see “Using DataWindow expression functions” on page 15.

Blanks examples

Assume City1 is "Austin" and City2 is " Austin ". Then the expression:

```
City1=City2
```

returns FALSE. PocketBuilder removes the trailing blank before making the comparison, but it does not remove the leading blank.

To prevent leading blanks from affecting a comparison, remove them with one of the Trim functions: Trim or LeftTrim.

For example:

```
Trim(City1)=Trim(City2)
```

returns TRUE.

To compare strings when trailing blanks are significant, use an expression such as the following to ensure that any trailing blanks are included in the comparison:

```
City1 + ">" = City2 + ">"
```

For information about these functions, see “Using DataWindow expression functions” on page 15.

Logical operators in DataWindow expressions

You use logical operators to combine boolean expressions into a larger boolean expression. The result is always TRUE or FALSE:

Table 1-9: Using expressions with logical operators

Operator	Meaning	Example
NOT	Logical negation. If A is true, NOT A is false. If A is false, NOT A is true.	NOT Price = 100
AND	Logical *and*. A AND B is true if both are true. A AND B is false if either is false.	Tax > 3 AND Ship < 5
OR	Logical *or*. A OR B is true if either is true or both are true. A OR B is false only if both are false.	Tax > 3 OR Ship < 5

When you combine two or more boolean expressions to form a new expression, the new expression is either true or false. The following truth table shows how TRUE and FALSE expressions are evaluated to form an expression that is either TRUE or FALSE.

For example, if "My dog has fleas" is true and "My hair is brown" is false, then "My dog has fleas OR my hair is brown" is true, and "My dog has fleas AND my hair is brown" is false:

Table 1-10: Combining expressions with logical operators

If one expression has this value	And the logical operator is	And if another expression has this value	The resulting expression has this value
TRUE	AND	TRUE	TRUE
TRUE	AND	FALSE	FALSE
FALSE	AND	TRUE	FALSE
FALSE	AND	FALSE	FALSE
TRUE	OR	TRUE	TRUE
TRUE	OR	FALSE	TRUE
FALSE	OR	TRUE	TRUE
FALSE	OR	FALSE	FALSE
NOT TRUE	AND	TRUE	FALSE
NOT TRUE	AND	FALSE	FALSE
NOT FALSE	AND	TRUE	TRUE
NOT FALSE	AND	FALSE	FALSE

If one expression has this value	And the logical operator is	And if another expression has this value	The resulting expression has this value
NOT TRUE	OR	TRUE	TRUE
NOT TRUE	OR	FALSE	FALSE
NOT FALSE	OR	TRUE	TRUE
NOT FALSE	OR	FALSE	TRUE

If you use a logical operator with a boolean function that returns NULL, the term with the NULL return value is evaluated as FALSE. If you use the NOT logical operator with a boolean function that returns NULL, the complete term evaluates to TRUE. For example, "NOT gf_boolean ()" evaluates to TRUE when gf_boolean () returns NULL.

Concatenation operator in DataWindow expressions

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

To concatenate values, you use the plus sign (+) operator.

Table 1-11: Using expressions with concatenation operator

String expression	Value
"over" + "stock"	overstock
Lname + ', ' + Fname	If Lname is Hill and Fname is Craig, then "Hill, Craig"

Using quotes
You can use either single or double quotes in string expressions. For example, the expression "over" + "stock" is equivalent to the expression `'over' + 'stock'`.

Operator precedence in DataWindow expressions

To ensure predictable results, operators in a DataWindow expression are evaluated in a specific order of precedence. When operators have the same precedence, they are evaluated from left to right.

The following table lists the operators in descending order of precedence:

Table 1-12: Operator precedence in DataWindow expressions

Operator	Purpose
( )	Grouping
^	Exponentiation
*, /	Multiplication and division
+, -	Addition and subtraction; string concatenation
IN,LIKE,BETWEEN	SQL SELECT statement conditions
=, >, <, <=, >=, <>	Relational operators
AND,OR	Logical *and* and logical *or*
NOT	Logical negation

Overriding the precedence order

Since expressions in parentheses are evaluated first, to override the precedence order, enclose expressions in parentheses. You can also use parentheses to clarify the order of evaluation. Within each set of parentheses, precedence order applies.

In the expression `x+y*a+b`, `y` is first multiplied by `a` (because multiplication has a higher precedence than addition). The result of the multiplication is then added to `x` and this result is then added to `b` (because the + operators are evaluated left to right).

To force evaluation in a different order, group expressions with parentheses. For example, in the expression `x+(y*(a+b))`, `a+b` is evaluated first. The sum `a+b` is then multiplied by `y`, and this product is added to `x`.

Evaluating DataWindow expressions in scripts

In a script, you use functions and data expressions for the DataWindow control to get information about the state of the DataWindow: the current row, the highlighted row, values of particular items. You can get other information by accessing properties of the DataWindow object, either with the Describe function or with property expressions.

For example, if you need to find the current row in a DataWindow, use the DataWindow control function, GetRow:

```
ll_rownum = dw_1.GetRow()
```

If you need to find the first row on the current page in a DataWindow, there is no function to return this information, but you can find it in the appropriate DataWindow object property:

```
ls_first = dw_1.Object.DataWindow.FirstRowOnPage
ls_last = dw_1.Object.DataWindow.LastRowOnPaged
w_1.Title = "Rows " + ls_first + " to " + ls_last
```

In some cases, however, information you need might not be available either by using DataWindow control functions or by accessing DataWindow object properties.

DataWindow expression functions sometimes provide information that is available in no other way. These functions, which are available within a DataWindow expression, are documented in "Using DataWindow expression functions" on page 15.

Evaluating DataWindow expressions in the Describe function

The Describe function provides a way to evaluate DataWindow expressions outside their usual context. The Evaluate function, which is used only within Describe, allows you to evaluate DataWindow expressions within a script using data in the DataWindow.

Evaluate has the following syntax:

dwcontrol .Describe ("Evaluate (*'expression'* , *rownumber*) ")

Expression is the expression you want to evaluate and rownumber is the number of the row for which you want to evaluate the expression. The expression can include DataWindow expression functions that cannot be called in a script.

This example displays in the title of the DataWindow control the current page for the current row in the DataWindow:

```
string ls_modstring, ls_rownum
ls_rownum = String(dw_1.GetRow())

ls_modstring = "Evaluate('Page()'," + ls_rownum +")"
// The resulting string, for row 99, would be:
// Evaluate('Page()', 99)

Parent.Title = &
"Current page: "+ dw_1.Describe(ls_modstring)
```

This example returns the display value for the dept_id column for row 5:

```
dw_1.Describe("Evaluate('LookUpDisplay(dept_id)', 5)")
```

Expressions that apply to all rows

To evaluate an expression that applies to all rows, specify 0 for the *rownumber* argument. This example calculates the sum of the salary column in the current DataWindow. It will return the expression's result or "!" if the expression is not valid:

```
dw_1.Describe("Evaluate('Sum(Salary)', 0)")
```

Evaluating user-specified expressions

In some types of applications, you might use Evaluate to get the result of an expression the user specifies. For example, users might specify the type of aggregation they want to see. This example evaluates an expression specified in a SingleLineEdit. It applies to all rows:

```
dw_1.Describe("Evaluate('" + sle_expr.Text + "', 0)")
```

Evaluating conditional DataWindow expressions with current data

Querying a property for a column

Values for column properties normally apply to all the rows in the column. For example, if you set the Protect property to "1" for the Emp_Id column, the user will be unable to modify the Emp_Id for any of the rows. If you query the property value for this column during execution, it will return "1".

When the column has a conditional expression

Instead of a constant, you can assign a conditional expression to some column properties. Properties are set on a row-by-row basis during execution.

For example, you might wish to allow users to enter an employee ID for new rows but protect this value for existing rows. The conditional expression for this column's Protect property would be:

```
If(IsRowNew(), 0, 1)
```

When you query the Protect property during execution, the result in this case would be the actual expression (preceded by a default value and a tab character and enclosed in quotes) instead of the property value. The value for the Protect property would be:

```
"0 <tab> If(IsRowNew(), 0, 1)"
```

Getting a property value for a particular row

To obtain the actual value of the Protect property for a particular row, you need to strip off the default value and the tab and evaluate the returned expression for the desired row. After stripping off the extra information, you can construct an expression for Describe that uses the Evaluate function.

This example checks whether the value of the Protect property for Emp_Id is a constant or a conditional expression. If it is a conditional expression, the script builds a string for the Describe function that uses Evaluate to get the value for of Protect for the current row:

```
string ls_protect, ls_eval
long ll_row

ll_row = dw_1.GetRow()
ls_protect = dw_1.Object.id.Protect

IF NOT IsNumber(ls_protect) THEN

       // Get the expression following the tab (~t)
       ls_protect = Right(ls_protect, &
          Len(ls_protect) - Pos(ls_protect, "~t"))

       // Build string for Describe. Include a leading
       // quote to match the trailing quote that remains
          ls_eval = "Evaluate(~"" + ls_protect + ", " &
          + String(ll_row) + ")"

       ls_protect = dw_1.Describe(ls_eval)

END IF

// Display result
st_result.Text = ls_protect
```

CHAPTER 2

DataWindow Expression Functions

About this chapter

This chapter provides syntax, descriptions, and examples of the functions you can use in expressions in the DataWindow painter.

Contents

After a short introduction and several examples, the functions are listed alphabetically.

Using DataWindow expression functions

In the DataWindow painter, you can use functions in expressions for computed fields, filters, validation rules, and graphed data, with some exceptions.

The dialog boxes in which you define expressions include a list box that lists the available functions and their arguments. The dialog boxes make it easy to insert a function into the expression.

For information about expressions, see Chapter 1, "DataWindow Operators and Expressions."

Return values for functions and expressions

DataWindow expressions can return the following datatypes:

- Double
- String
- DateTime
- Time

Within an expression, a function can return other datatypes (such as boolean, date, or integer), but the final value of an expression is converted to one of the four datatypes.

Restrictions for aggregate functions

An aggregate function is a function (such as Avg, Max, StDev, and Sum) that operates on a range of values in a column. When you use an aggregate function, some restrictions apply. You cannot use an aggregate function:

- In a filter
- In a validation rule
- As an argument for another aggregate function

When you use aggregate functions, they cancel the effect of setting Retrieve Rows As Needed. To do the aggregation, the DataWindow object always retrieves all rows.

User-defined functions in PocketBuilder

You can include user-defined functions in DataWindow expressions. The datatype of the function's return value can be any of the following: double, string, boolean, date, DateTime, or time. The function must be defined as a global function so that it is available to the DataWindow object.

Built-in DataWindow expression functions 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.

Formatting for the locally correct display of numbers

No matter what country you are creating objects and developing an application in, you must use U.S. number notation in numbers or number masks in display formats, edit masks, and DataWindow expressions. This means that when you specify a number or number mask, use a comma as the thousands delimiter and period for the decimal place.

Numbers display appropriately in whatever countries you deploy applications in. During execution, the locally correct symbols for numbers display (because the international Control Panel settings are used) when numbers are interpreted. For example, in countries where comma represents the decimal place and period represents thousands, users see numbers in those formats during execution.

For information about the locally correct display of dates and day names, see String on page 111 and DayName on page 46.

Four examples

Example 1: counting NULL values in a column

A NULL value is a marker used to fill a place in a column where data is missing for any reason. The value might not be applicable, or it might be missing or unknown. When a database table is created, each column in the table either allows NULL values or does not allow them. The column or set of columns that define the primary key cannot allow NULL values. Sometimes it is useful to know how many NULL values there are in a particular column.

What you want to do

You are working with the Fin_code table in the Enterprise Application Sample Database. The Fin_code table has three columns:

Table 2-1: Columns in the Fin_code table

Column	What the column is	Allows NULL values?
Code	Unique financial identifier (primary key)	No
Type	Code type: expense or revenue	No
Description	Code description: the department incurring the expense or getting the revenue	Yes

You create a DataWindow object using the Code and Description columns. You want to know the number of NULL values in the Description column.

How to do it

In the DataWindow object, you create a computed field that uses functions to display the number of NULL values in the Description column.

For the sake of demonstrating the use of functions, the following computed fields are created in the Summary band of the DataWindow object (with text objects that tell you what information each computed field is providing):

```
Count(description for all)
```

which counts the number of descriptions (that are not NULL);

```
Sum(If(IsNull(description), 1, 0))
```

which returns a 1 if the description column is NULL, a 0 if the description column is NOT NULL, and then adds the total;

```
Count(id for all)
```

which counts the number of IDs (which is also the number of rows);

```
Sum(If(IsNull(description), 1, 1))
```

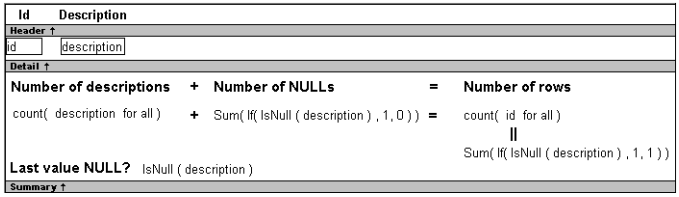
which adds the number of NULLs and NOT NULLs in the description column (which is the total number of rows) and should match the result of the Count(id for all) function; and

```
IsNull(description)
```

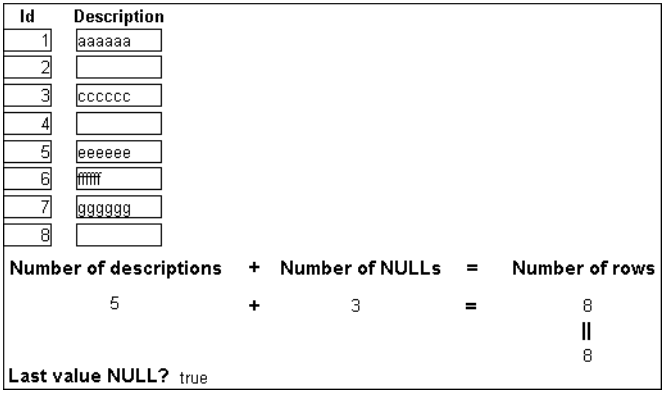
which evaluates whether the last row in the table has a description that is NULL. The return value of the IsNull function is TRUE or FALSE.

What you get

Here is the design for the DataWindow object.



Here is the DataWindow object showing eight descriptions, three of which are NULL and five of which are not NULL. The last description for Id=8 is NULL.



Example 2: counting male and female employees

Example 1 demonstrates the use of the Sum and Count functions. Sum and Count are two examples of a class of functions called aggregate functions.

An aggregate function is a function that operates on a range of values in a column. The aggregate functions are:

Avg	Large	Mode	Sum
Count	Last	Percent	Var
CumulativePercent	Max	Small	VarP
CumulativeSum	Median	StDev	
First	Min	StDevP	

About crosstab functions
Although the crosstab functions (CrosstabAvg, CrosstabCount, CrosstabMax, CrosstabMin, and CrosstabSum) behave like aggregate functions, they are not included on the list because they are for crosstabs only and are designed to work in the crosstab matrix.

A few restrictions apply to the use of aggregate functions. You cannot use an aggregate function:

- In a filter
- In a validation rule
- As an argument for another aggregate function

This example demonstrates the use of the Sum aggregate function.

What you want to do

Using the Employee table in the Enterprise Application Sample Database as the data source, you create a DataWindow object using at least the Emp_id and the Sex columns. You want the DataWindow object to display the number of male employees and female employees in the company.

How to do it

In the summary band in the workspace, add two computed fields to the DataWindow object that use the Sum and If functions:

```
Sum(If(sex = "M", 1, 0))
```

counts the number of males in your company;

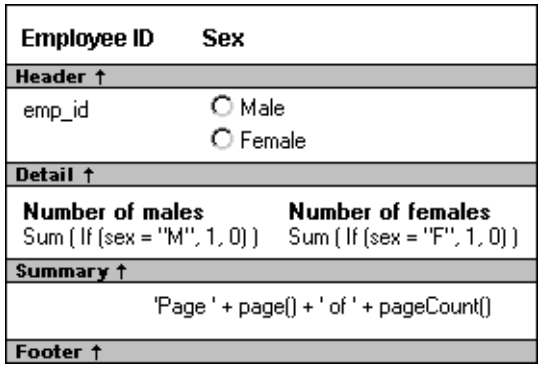
```
Sum(If(sex = "F", 1, 0))
```

counts the number of females in your company.

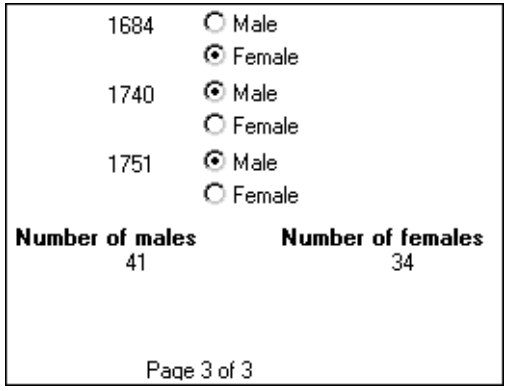
You can also add a Page computed field (by clicking the Page computed field button) in the footer band to display the page number and total pages at the bottom of each page of the DataWindow object.

What you get

Here is what the design of the DataWindow object looks like.



Here is the last page of the DataWindow object, with the total number of males and females in the company displayed.



If you now want more information

What if you decide that you also want to know the number of males and females in each department in the company?

❖ **To display the males and females in each department:**

1 Select Design>Data Source from the menu bar so that you can edit the data source.

2 Select Design>Select tables from the menu bar and open the Department table in the Select painter workspace, which currently displays the Employee table with the Emp_id and Sex columns selected.

3 Select the department_dept_name column to add it to your data source.

4 Select Rows>Create Group from the menu bar to create a group and group by department name.

5 In the trailer group band, add two additional computed fields:

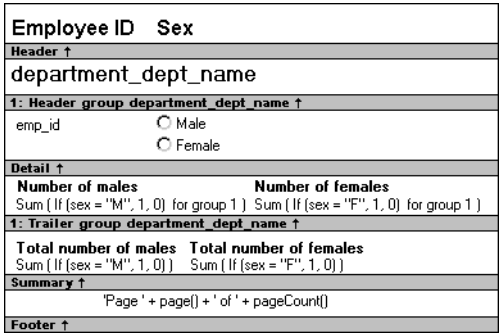
```
Sum(If(sex = "M", 1, 0) for group 1)
```

counts the number of males in each department;

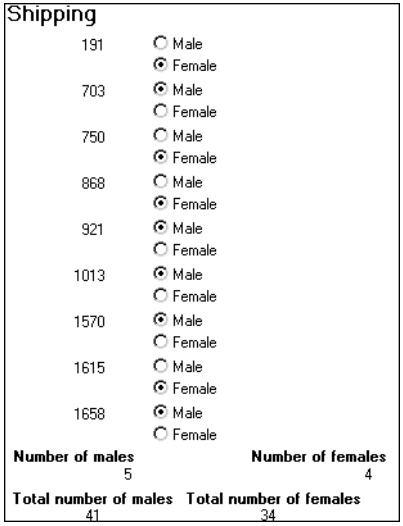
```
Sum(If(sex = "F", 1, 0) for group 1)
```

counts the number of females in each department.

Here is what the design of the grouped DataWindow object looks like.



Here is the last page of the DataWindow object with the number of males and females in the shipping department displayed, followed by the total number of males and females in the company.



Example 3: creating a row indicator

This example demonstrates the use of several functions: Bitmap, Case, CurrentRow, GetRow, and RGB.

What you want to do

Using the Employee table in the Enterprise Application Sample Database, you create a DataWindow object using the Emp_id, Emp_fname, Emp_lname, and Salary columns.

In the DataWindow painter, you want to display a number of items such as the number of the current row, an arrow that is an indicator of the current row, and the salary for an employee with a background color that depends on what the salary is.

How to do it

In the workspace, add the following:

- A computed field CurrentRow( ), which displays the number of the current row
- A picture object, which is a right-arrow, for which you define an expression for the arrow's visible property:

  ```
  If(CurrentRow()= GetRow(), 1, 0)
  ```

  The expression causes an arrow to display in the current row and no arrow to display in other rows.
- A computed field using the If, CurrentRow, and GetRow functions:

  ```
  If(CurrentRow() = GetRow(),"Current","Not current")
  ```

  which displays the word "Current" when the row is the current row and "Not current" for all other rows
- A computed field (typed on one line) using the Bitmap, CurrentRow, and GetRow functions:

  ```
  Bitmap(If(CurrentRow()= GetRow(),
  "c:\sampl\ex\code\indicatr.bmp", " "))
  ```

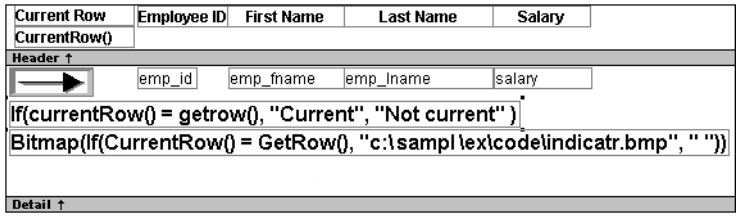
  which displays an arrow bitmap for the current row and no bitmap for all other rows
- An expression for the Background.Color property of the salary column:

  ```
  Case(salary WHEN IS >60000 THEN RGB(192,192,192)
  WHEN IS >40000 THEN RGB(0,255,0) ELSE
  RGB(255,255,255))
  ```

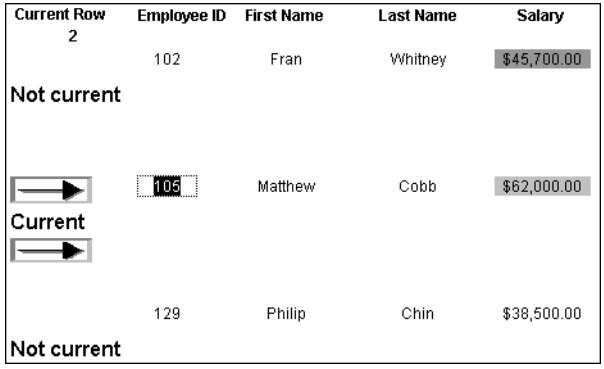
  The expression causes a salary above $40,000 to display in green, a salary above $60,000 to display in gray, and all other salaries to display in white.

What you get

Here is what the design of the DataWindow object looks like:



Here is what the data looks like with the second row current.



Notice that the number of the current row is 2; the first row and the third row are "Not current" (and therefore display no bitmap); and the second row, which is the current row, displays the arrow row indicator.

On your screen, the salary in the first row has a green background because it is more than $40,000; the salary in the second row has a gray background because it is more than $60,000; and the salary in the third row has a white background, which matches the background of the DataWindow object.

Example 4: displaying all data when a column allows NULLs

When you create an arithmetic expression that has a NULL value, the value of the expression is NULL. This makes sense, since NULL means essentially undefined and the expression is undefined, but sometimes this fact can interfere with what you want to display.

What you want to do

A table in your database has four columns: Id, Corporation, Address1, and Address2. The Corporation, Address1, and Address2 columns allow NULLs. Using this table as the data source, you create a DataWindow object using the four columns. You now want the DataWindow object to display both parts of the address, separated by a comma.

You create a computed field to concatenate Address1 and Address2 with a comma separator. Here is the expression that defines the computed field:

```
address1 + ", " + address2
```

When you preview the DataWindow object, if either Address1 or Address2 is NULL, no part of the address displays because the value of the expression is NULL. To display a part of the address, you need to create a computed field that forces evaluation even if Address2 is NULL. Note that Address2 is assumed to have data only if Address1 has data for a particular row.

How to do it

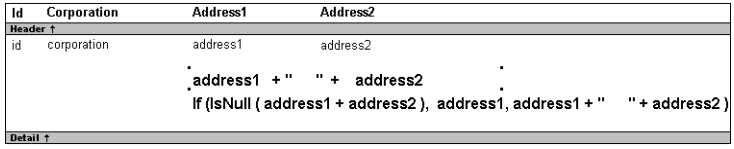
In the detail band, create a computed field that uses the If and IsNull functions:

```
If(IsNull(address1 + address2), address1, address1
+ ", " + address2)
```

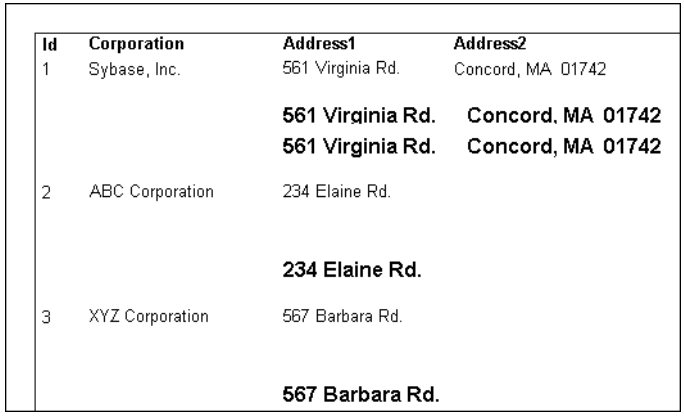
The computed field says this: if the concatenation of the addresses is NULL (because address2 is NULL), then display address1, and if it is not NULL, display both parts of the address separated by a comma.

What you get

Here is what the design of the DataWindow object looks like. It includes both the computed field that does not work and the one that does.



When you preview the DataWindow object, notice that the first computed field displays NULL for ABC Corporation and XYZ Corporation. The second computed field displays the first part of the address, which is not NULL.



Id	Corporation	Address1	Address2
1	Sybase, Inc.	561 Virginia Rd.	Concord, MA 01742
		561 Virginia Rd.	**Concord, MA 01742**
		561 Virginia Rd.	**Concord, MA 01742**
2	ABC Corporation	234 Elaine Rd.	
		234 Elaine Rd.	
3	XYZ Corporation	567 Barbara Rd.	
		567 Barbara Rd.	

Alphabetical list of DataWindow expression functions

The list of DataWindow expression functions follows in alphabetical order.

Abs

Description

Calculates the absolute value of a number.

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*.

Examples

This expression counts all the product numbers where the absolute value of the product number is distinct:

```
Count(product_number for All DISTINCT Abs
(product_number))
```

Only data with an absolute value greater than 5 passes this validation rule:

```
Abs(value_set) > 5
```

See also

Count
Abs in the *PowerScript Reference*

ACos

Description

Calculates the arccosine of an angle.

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* if it succeeds.

Examples

This expression returns 0:

```
ACos(1)
```

This expression returns 3.141593 (rounded to six places):

```
ACos(-1)
```

This expression returns 1.000000 (rounded to six places):

```
ACos(.540302)
```

See also

Cos
ASin
ATan
ACos in the *PowerScript Reference*

Asc

Description

Converts the first character of a string to its ASCII integer value.

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*.

Usage

Use Asc to test the case of a character or manipulate text and letters.

To find out the case of a character, you can check whether its ASCII value is within the appropriate range.

Examples

This expression for a computed field returns the string in code_id if the ASCII value of the first character in code_id is A (65):

```
If (Asc(code_id) = 65, code_id, "Not a valid code")
```

This expression for a computed field checks the case of the first character of lname and if it is lowercase, makes it uppercase:

```
IF (Asc(lname) > 64 AND Asc(lname) < 91, lname,
WordCap(lname))
```

See also

Char
WordCap
Asc in the *PowerScript Reference*

ASin

Description

Calculates the arcsine of an angle.

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* if it succeeds.

Examples

This expression returns .999998 (rounded to six places):

```
ASin(.84147)
```

This expression returns .520311 (rounded to six places):

```
ASin(LogTen (Pi (1)))
```

This expression returns 0:

```
ASin(0)
```

See also

Sin
ACos
ATan
Pi
ASin in the *PowerScript Reference*

ATan

Description

Calculates the arctangent of an angle.

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* if it succeeds.

Examples

This expression returns 0:

```
ATan(0)
```

This expression returns 1.000 (rounded to three places):

```
ATan(1.55741)
```

This expression returns 1.267267 (rounded to six places):

```
ATan(Pi(1))
```

See also

Tan
ASin
ACos
ATan in the *PowerScript Reference*

Avg

Description

Calculates the average of the values of the column.

Syntax

Avg (*column* { FOR *range* { DISTINCT { *expres1* {, *expres2* {, ... } } } } })

Argument	Description
column	The column for which you want the average of the data values. *Column* can be the column name or the column number preceded by a pound sign (#). *Column* can also be an expression that includes a reference to the column. The datatype of *column* must be numeric.
FOR *range* (optional)	The data that will be included in the average. For most presentation styles, values for *range* are: • ALL — (Default) The average of all values in *column.* • GROUP *n* — The average of values in *column* in the specified group. Specify the keyword GROUP followed by the group number: for example, GROUP 1. • PAGE — The average of the values in *column* on a page. For Graph objects, specify one of the following: • GRAPH — The average of values in *column* in the range specified for the Rows option.
DISTINCT (optional)	Causes Avg to consider only the distinct values in *column* when calculating the average. For a value of *column*, the first row found with the value is used and other rows that have the same value are ignored.
expresn (optional)	One or more expressions that you want to evaluate to determine distinct rows. *Expresn* can be the name of a column, a function, or an expression.

Return value

The numeric datatype of the column. Returns the average of the values of the rows in *range*.

Usage

If you specify *range*, Avg returns the average value of *column* in *range*. If you specify DISTINCT, Avg returns the average value of the distinct values in *column*, or if you specify *expresn*, the average of *column* for each distinct value of *expresn*.

For graphs, you do not select the range when you call the function. The range has already been determined by the Rows setting on the Data property page (the Range property), and the aggregation function uses that range. Settings for Rows include the following:

- For the Graph presentation style, Rows is always All.
- For Graph controls, Rows can be All, Page, or Group.

In calculating the average, NULL values are ignored.

Not in validation rules or filter expressions
You cannot use this or other aggregate functions in validation rules or filter expressions.

Using an aggregate function cancels the effect of setting Retrieve Rows As Needed in the painter. To do the aggregation, a DataWindow object or report always retrieves all rows.

Examples

This expression returns the average of the values in the column named salary:

```
Avg(salary)
```

This expression returns the average of the values in group 1 in the column named salary:

```
Avg(salary for group 1)
```

This expression returns the average of the values in column 5 on the current page:

```
Avg(#5 for page)
```

This computed field returns Above Average if the average salary for the page is greater than the average salary:

```
If(Avg(salary for page) > Avg(salary), "Above Average",
" ")
```

This expression for a graph value sets the data to the average value of the sale_price column:

```
Avg(sale_price)
```

This expression for a graph value sets the data value to the average value of the sale_price column for the entire graph:

```
Avg(sale_price for graph)
```

Assuming a DataWindow object displays the order number, amount, and line items for each order, this computed field returns the average of the order amount for the distinct order numbers:

```
Avg(order_amt for all DISTINCT order_nbr)
```

See also

Median
Mode

Bitmap

Description

Displays the specified bitmap.

For computed fields only
You can use the Bitmap function *only* in a computed field.

Syntax

Bitmap (*string*)

Argument	Description
string	A column containing bitmap files, a string containing the name of an image file (a BMP, GIF, or JPEG file), or an expression that evaluates to a string containing the name of an image file

Return value

The special datatype bitmap, which *cannot* be used in any other function.

Usage

Use Bitmap to dynamically display a bitmap in a computed field. When *string* is a column containing bitmap files, a different bitmap can display for each row.

Examples

These examples are all expressions for a computed field.

This expression dynamically displays the bitmap file contained in the column named employees:

```
Bitmap(employees)
```

If the employees column is column 3, this next expression gives the same result as the expression above:

```
Bitmap(#3)
```

This expression displays the bitmap TOOLS.BMP:

```
Bitmap("TOOLS.BMP")
```

This expression tests the value in the column named password and then uses the value to determine which bitmap to display:

```
Bitmap(If(password = "y", "yes.bmp", "no.bmp"))
```

See also "Example 3: creating a row indicator" on page 22

Case

Description Tests the values of a column or expression and returns values based on the results of the test.

Syntax **Case** (*column* WHEN *value1* THEN *result1* { WHEN *value2* THEN *result2* { ... } } { ELSE *resultelse* })

Argument	Description
column	The column or expression whose values you want to test. *Column* can be the column name or the column number preceded by a pound sign (#). *Column* can also be an expression that includes a reference to the column. *Column* is compared to each *valuen*.
WHEN (optional)	Introduces a value-result pair. At least one WHEN is required.
valuen	One or more values that you want to compare to values of *column*. A value can be: • A single value • A list of values separated by commas (for example, 2, 4, 6, 8) • A TO clause (for example, 1 TO 20) • IS followed by a relational operator and comparison value (for example, IS>5) • Any combination of the above with an implied OR between expressions (for example, 1,3,5,7,9,27 TO 33, IS>42)
THEN	Introduces the result to be returned when *column* matches the corresponding *valuen*.
resultn	An expression whose value is returned by Case for the corresponding *valuen*. All *resultn* values must have the same datatype.
ELSE (optional)	Specifies that for any values of *column* that do not match the values of *valuen* already specified, Case returns *resultelse*.
resultelse	An expression whose value is returned by Case when the value of *column* does not match any WHEN *valuen* expression.

Return value The datatype of *resultn*. Returns the result you specify in *resultn*.

Usage

If more than one WHEN clause matches *column*, Case returns the result of the first matching one.

Examples

This expression for the Background.Color property of a Salary column returns values that represent red when an employee's salary is greater than $70,000, green when an employee's salary is greater than $50,000, and blue otherwise:

```
Case(salary WHEN IS >70000 THEN RGB(255,0,0) WHEN IS
>50000 THEN RGB(0,255,0) ELSE RGB(0,0,255))
```

This expression for the Background.Color property of an employee Id column returns red for Id 101, gray for Id 102, and black for all other Id numbers:

```
Case(emp_id WHEN 101 THEN 255 WHEN 102 THEN
RGB(100,100,100) ELSE 0)
```

This expression for the Format property of the Marital_status column returns Single, Married, and Unknown based on the data value of the Marital_status column for an employee:

```
Case(marital_status WHEN 'S'THEN 'Single' WHEN 'M' THEN
'Married' ELSE 'Unknown')
```

See also

"Example 3: creating a row indicator" on page 22
If

Ceiling

Description

Retrieves the smallest whole number that is greater than or equal to a specified limit.

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*.

Examples

These expressions both return - 4:

```
Ceiling(-4.2)

Ceiling(-4.8)
```

This expression for a computed field returns ERROR if the value in discount_amt is greater than the smallest whole number that is greater than or equal to discount_factor times price. Otherwise, it returns discount_amt:

```
If(discount_amt <= Ceiling(discount_factor * price),
String(discount_amt), "ERROR")
```

To pass this validation rule, the value in discount_amt must be less than or equal to the smallest whole number that is greater than or equal to discount_factor times price:

```
discount_amt <= Ceiling(discount_factor * price)
```

See also

Int
Round
Truncate
Ceiling in the *PowerScript Reference*

Char

Description

Converts an integer to a character.

Syntax

Char (*n*)

Argument	Description
n	The integer you want to convert to a character

Return value

String. Returns the character whose ASCII value is *n*.

Examples

This expression returns the escape character:

```
Char(27)
```

See also

Asc
Char in the *PowerScript Reference*

Cos

Description

Calculates the cosine of an angle.

Syntax

Cos (*n*)

Argument	Description
n	The angle (in radians) for which you want the cosine

Return value

Double. Returns the cosine of *n*.

Examples

This expression returns 1:

```
Cos(0)
```

This expression returns .540302:

```
Cos(1)
```

This expression returns - 1:

```
Cos(Pi(1))
```

See also

Pi
Sin
Tan
Cos in the *PowerScript Reference*

Count

Description

Calculates the total number of rows in the specified column.

Syntax

Count (*column* { FOR *range* { DISTINCT { *expres1* {, *expres2* {, ... } } } } })

Argument	Description
column	The column for which you want the number of rows. *Column* can be the column name or the column number preceded by a pound sign (#). *Column* can also be an expression that includes a reference to the column.
FOR *range* (optional)	The data that will be included in the count. For most presentation styles, values for *range* are: • ALL — (Default) The count of all rows in *column*. • GROUP *n* — The count of rows in *column* in the specified group. Specify the keyword GROUP followed by the group number: for example, GROUP 1. • PAGE — The count of the rows in *column* on a page. For Graph objects, specify one of the following: • GRAPH — The count of values in *column* in the range specified for the Rows option.

Argument	Description
DISTINCT (optional)	Causes Count to consider only the distinct values in *column* when counting the rows. For a value of *column*, the first row found with the value is used and other rows that have the same value are ignored.
expresn (optional)	One or more expressions that you want to evaluate to determine distinct rows. *Expresn* can be the name of a column, a function, or an expression.

Usage

If you specify *range*, Count determines the number of rows in *column* in *range*. If you specify DISTINCT, Count returns the number of the distinct rows displayed in *column*, or if you specify *expresn*, the number of rows displayed in *column* where the value of *expresn* is distinct.

For graphs, you do not select the range when you call the function. The range has already been determined by the Rows setting on the Data property page (the Range property), and the aggregation function uses that range.

Settings for Rows include the following:

- For the Graph presentation style, Rows is always All.
- For Graph controls, Rows can be All, Page, or Group.

Null values in the column are ignored and are not included in the count.

Not in validation rules or filter expressions
You cannot use this or other aggregate functions in validation rules or filter expressions.

Using an aggregate function cancels the effect of setting Retrieve Rows As Needed in the painter. To do the aggregation, a DataWindow object or a report always retrieves all rows.

Examples

This expression returns the number of rows in the column named emp_id that are not NULL:

```
Count(emp_id)
```

This expression returns the number of rows in the column named emp_id of group 1 that are not NULL:

```
Count(emp_id for group 1)
```

This expression returns the number of dept_ids that are distinct:

```
Count(dept_id for all DISTINCT)
```

This expression returns the number of regions with distinct products:

```
Count(region_id for all DISTINCT Lower(product_id))
```

This expression returns the number of rows in column 3 on the page that are not NULL:

```
Count(#3 for page)
```

See also "Example 1: counting NULL values in a column" on page 17

CrosstabAvg

Description Calculates the average of the values returned by an expression in the values list of the crosstab. When the crosstab definition has more than one column, CrosstabAvg can also calculate averages of the expression's values for groups of column values.

PocketBuilder
This function is not available for DataWindow objects that you use in PocketBuilder. You can use this function in a crosstab DataWindow object or report only.

Syntax **CrosstabAvg** (*n* {, *column*, *groupvalue* })

Return value Double. Returns the average of the crosstab values returned by expression *n* for all the column values or, optionally, for a subset of column values.

CrosstabCount

Description Counts the number of values returned by an expression in the values list of the crosstab. When the crosstab definition has more than one column, CrosstabCount can also count the number of the expression's values for groups of column values.

PocketBuilder
This function is not available for DataWindow objects that you use in PocketBuilder. You can use this function in a crosstab DataWindow object or report only.

Syntax **CrosstabCount** (*n* {, *column*, *groupvalue* })

Return value

Long. Returns the number of values returned by expression *n* for all the column values or, optionally, for a subset of column values.

CrosstabMax

Description

Calculates the maximum value returned by an expression in the values list of the crosstab. When the crosstab definition has more than one column, CrosstabMax can also calculate the maximum of the expression's values for groups of column values.

PocketBuilder
This function is not available for DataWindow objects that you use in PocketBuilder. You can use this function in a crosstab DataWindow object or report only.

Syntax

CrosstabMax (*n* {, *column*, *groupvalue* })

Return value

Double. Returns the maximum value returned by expression *n* for all the column values or, optionally, for a subset of column values.

CrosstabMin

Description

Calculates the minimum value returned by an expression in the values list of the crosstab. When the crosstab definition has more than one column, CrosstabMin can also calculate the minimum of the expression's values for groups of column values.

PocketBuilder
This function is not available for DataWindow objects that you use in PocketBuilder. You can use this function in a crosstab DataWindow object or report only.

Syntax

CrosstabMin (*n* {, *column*, *groupvalue* })

Return value

Double. Returns the minimum value returned by expression *n* for all the column values or, optionally, for a subset of column values.

CrosstabSum

Description

Calculates the sum of the values returned by an expression in the values list of the crosstab. When the crosstab definition has more than one column, CrosstabSum can also calculate the sum of the expression's values for groups of column values.

PocketBuilder
This function is not available for DataWindow objects that you use in PocketBuilder. You can use this function in a crosstab DataWindow object or report only.

Syntax

CrosstabSum (*n* {, *column*, *groupvalue* })

Return value

Double. Returns the total of the values returned by expression *n* for all the column values or, optionally, for a subset of column values.

CumulativePercent

Description

Calculates the total value of the rows up to and including the current row in the specified column as a percentage of the total value of the column (a running percentage).

Syntax

CumulativePercent (*column* { FOR *range* })

Argument	Description
column	The column for which you want the cumulative value of the rows up to and including the current row as a percentage of the total value of the column for *range*. *Column* can be the column name or the column number preceded by a pound sign (#). *Column* can also be an expression that includes a reference to the column. The datatype of *column* must be numeric.
FOR *range* (optional)	The data that will be included in the cumulative percentage. For most presentation styles, values for *range* are: • ALL — (Default) The cumulative percentage of all rows in *column*. • GROUP *n* — The cumulative percentage of rows in *column* in the specified group. Specify the keyword GROUP followed by the group number: for example, GROUP 1. • PAGE — The cumulative percentage of the rows in *column* on a page. For Graph objects, specify one of the following: • GRAPH — The cumulative percentage of values in *column* in the range specified for the Rows option.

Return value

Long. Returns the cumulative percentage value.

Usage

If you specify *range*, CumulativePercent restarts the accumulation at the start of the range.

For graphs, you do not select the range when you call the function. The range has already been determined by the Rows setting on the Data property page (the Range property), and the aggregation function uses that range.

Settings for Rows include the following:

- For the Graph presentation style, Rows is always All.
- For Graph controls, Rows can be All, Page, or Group.

In calculating the percentage, NULL values are ignored.

Not in validation rules or filter expressions
You cannot use this or other aggregate functions in validation rules or filter expressions.

Using an aggregate function cancels the effect of setting Retrieve Rows As Needed in the painter. To do the aggregation, a DataWindow object or a report always retrieves all rows.

Examples

This expression returns the running percentage for the values that are not NULL in the column named salary:

```
CumulativePercent(salary)
```

This expression returns the running percentage for the column named salary for the values in group 1 that are not NULL:

```
CumulativePercent(salary for group 1)
```

This expression entered in the Value box on the Data property page for a graph returns the running percentage for the salary column for the values in the graph that are not NULL:

```
CumulativePercent(salary for graph)
```

See also

Percent
CumulativeSum

CumulativeSum

Description

Calculates the total value of the rows up to and including the current row in the specified column (a running total).

Syntax

CumulativeSum (*column* { FOR *range* })

Argument	Description
column	The column for which you want the cumulative total value of the rows up to and including the current row for group. *Column* can be the column name or the column number preceded by a pound sign (#). *Column* can also be an expression that includes a reference to the column. The datatype of *column* must be numeric.
FOR *range* (optional)	The data that will be included in the cumulative sum. For most presentation styles, values for *range* are: • ALL — (Default) The cumulative sum of all values in *column*. • GROUP *n* — The cumulative sum of values in *column* in the specified group. Specify the keyword GROUP followed by the group number: for example, GROUP 1. • PAGE — The cumulative sum of the values in *column* on a page. For Graph objects, specify one of the following: • GRAPH — The cumulative sum of values in *column* in the range specified for the Rows option.

Return value

Long. Returns the cumulative total value of the rows.

Usage

If you specify *range*, CumulativeSum restarts the accumulation at the start of the range.

For graphs, you do not select the range when you call the function. The range has already been determined by the Rows setting on the Data property page (the Range property), and the aggregation function uses that range. Settings for Rows include the following:

- For the Graph presentation style, Rows is always All.
- For Graph controls, Rows can be All, Page, or Group.

In calculating the sum, NULL values are ignored.

Examples

This expression returns the running total for the values that are not NULL in the column named salary:

```
CumulativeSum(salary)
```

This expression returns the running total for the values that are not NULL in the column named salary in group 1:

```
CumulativeSum(salary for group 1)
```

This expression returns the running total for the values that are not NULL in the column named salary in group 1:

```
CumulativeSum(salary for group 1)
```

This expression entered in the Value box on the Data property page for a graph returns the running total for the salary column for the values in the graph that are not NULL:

```
CumulativeSum(salary for graph)
```

See also

CumulativePercent

CurrentRow

Description

Reports the number of the current row (the row with focus).

Syntax

CurrentRow ( )

Return value

Long. Returns the number of the row if it succeeds and 0 if no row is current.

What row is current
The current row is not always a row displayed on the screen. For example, if the cursor is on row 7 column 2 and the user uses the scroll bar to scroll to row 50, the current row remains row 7 unless the user clicks row 50.

Examples

This expression in a computed field returns the number of the current row:

```
CurrentRow()
```

This expression for a computed control displays an arrow bitmap as an indicator for the row with focus and displays no bitmap for rows not having focus. As the user moves from row to row, an arrow marks where the user is:

```
Bitmap(If(CurrentRow() = GetRow(),"arrow.bmp",""))
```

Alternatively, this expression for the Visible property of an arrow picture control makes the arrow bitmap visible for the row with focus and invisible for rows not having focus. As the user moves from row to row, an arrow marks where the user is:

```
If(CurrentRow() = GetRow(), 1, 0)
```

See also

"Example 3: creating a row indicator" on page 22
GetRow

Date

Description

Converts a string whose value is a valid date to a value of datatype date.

Syntax

Date (*string*)

Argument	Description
string	A string containing a valid date (such as Jan 1, 1998, or 12-31-99) that you want returned as a date

Return value

Date. Returns the date in *string* as a date. If *string* does not contain a valid date, Date returns NULL.

Regional 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 to have the application change it (by calling the RegistrySet function). The user might need to reboot after the setting is changed.

Usage

The value of the string must be a valid date.

Valid dates
Valid dates can include any combination of day (1–31), month (1–12 or the name or abbreviation of a month), and year (two or four digits). Leading zeros are optional for month and day. If the month is a name or an abbreviation, it can come before or after the day; if it is a number, it must be in the month location specified in the Windows control panel. A 4-digit number is assumed to be a year.

If the year is two digits, the assumption of century follows this rule: for years between 00 and 49, the first two digits are assumed to be 20; for years between 50 and 99, the first two digits are assumed to be 19. If your data includes dates before 1950, such as birth dates, always specify a four-digit year to ensure the correct interpretation.

Years from 1000 to 3000 inclusive are handled.

An expression has a more limited set of datatypes than the functions that can be part of the expression. Although the Date function returns a date value, the whole expression is promoted to a DateTime value. Therefore, if your expression consists of a single Date function, it will appear that Date returns the wrong datatype. To display the date without the time, choose an appropriate display format. (See "Using DataWindow expression functions" on page 15.)

Examples

These expressions all return the date datatype for July 4, 1999 when the default location of the month in Regional Settings is center:

```
Date("1999/07/04")
Date("1999 July 4")
Date("July 4, 1999")
```

See also

IsDate
Date in the *PowerScript Reference*

DateTime

Description

Combines a date and a time value into a DateTime value.

Syntax

DateTime (*date* {, *time* })

Argument	Description
date	A valid date (such as Jan 1, 1998, or 12-31-99) or a blob variable whose first value is a date that you want included in the value returned by DateTime.
time (optional)	A valid time (such as 8am or 10:25:23:456799) or a blob variable whose first value is a time that you want included in the value returned by DateTime. If you include a time, only the hour portion is required. If you omit the minutes, seconds, or microseconds, they are assumed to be zeros. If you omit am or pm, the hour is determined according to the 24-hour clock.

Return value

DateTime. Returns a DateTime value based on the values in *date* and optionally *time*. If time is omitted, DateTime uses 00:00:00.000000 (midnight).

Usage

To display microseconds in a time, the display format for the field must include microseconds.

For information on valid dates, see Date.

Examples

This expression returns the values in the order_date and order_time columns as a DateTime value that can be used to update the database:

```
DateTime(Order_Date, Order_Time)
```

Using this expression for a computed field displays `11/11/01 11:11:00`:

```
DateTime(11/11/01, 11:11)
```

See also

Date
Time
DateTime in the *PowerScript Reference*

Day

Description

Obtains the day of the month in a date value.

Syntax

Day (*date*)

Argument	Description
date	The date for which you want the day

Return value

Integer. Returns an integer (1–31) representing the day of the month in *date*.

Examples

This expression returns 31:

```
Day(1999-01-31)
```

This expression returns the day of the month in the start_date column:

```
Day(start_date)
```

See also

Date
IsDate
Month
Year
Day in the *PowerScript Reference*

DayName

Description

Gets the day of the week in a date value and returns the weekday's name.

Syntax

DayName (*date*)

Argument	Description
date	The date for which you want the name of the day

Return value

String. Returns a string whose value is the name of the weekday (Sunday, Monday, and so on) for *date*.

Usage

DayName returns a name in the language of the deployment files available on the machine where the application is run. If you have installed localized deployment files in the development environment or on a user's machine, then on that machine the name returned by DayName will be in the language of the localized files.

Examples

This expression for a computed field returns Okay if the day in date_signed is not Sunday:

```
If(DayName(date_signed) <> "Sunday", "Okay", "Invalid
Date")
```

To pass this validation rule, the day in date_signed must not be Sunday:

```
DayName(date_signed) <> "Sunday"
```

See also

Date
Day
DayNumber
IsDate
DayName in the *PowerScript Reference*

DayNumber

Description

Gets the day of the week of a date value and returns the number of the weekday.

Syntax

DayNumber (*date*)

Argument	Description
date	The date 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.

Examples

This expression for a computed field returns Wrong Day if the date in start_date is not a Sunday or a Monday:

```
If(DayNumber(start_date) > 2, "Okay", "Wrong Day")
```

This expression for a computed field returns Wrong Day if the date in end_date is not a Saturday or a Sunday:

```
If(DayNumber(end_date) > 1 and DayNumber(end_date)
< 7, "Okay", "Wrong Day")
```

This validation rule for the column end_date ensures that the day is not a Saturday or Sunday:

```
DayNumber(end_date) >1 and DayNumber(end_date) < 7
```

See also

Date
Day
DayName
IsDate
DayNumber in the *PowerScript Reference*

DaysAfter

Description

Gets the number of days one date occurs after another.

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 containing the number of days *date2* occurs after *date1*. If *date2* occurs before *date1*, DaysAfter returns a negative number.

Examples

This expression returns 4:

```
DaysAfter(1999-12-20, 1999-12-24)
```

This expression returns -4:

```
DaysAfter(1999-12-24, 1999-12-20)
```

This expression returns 0:

```
DaysAfter(1999-12-24, 1999-12-24)
```

This expression returns 5:

```
DaysAfter(1998-12-29, 1999-01-03)
```

See also

Date
SecondsAfter
DaysAfter in the *PowerScript Reference*

Describe

Description

Reports the values of properties of a DataWindow object and controls within the object. Each column and graphic control in the DataWindow object has a set of properties, which are listed in “Controls in a DataWindow and their properties” on page 128. You specify one or more properties as a string and Describe returns the values of the properties.

Syntax

Describe (*propertylist*)

Argument	Description
propertylist	A string whose value is a blank-separated list of properties or Evaluate functions. For a list of valid properties, see “Controls in a DataWindow and their properties” on page 128.

Return value

String. Returns a string that includes a value for each property or Evaluate function. A newline character (~n) separates the value of each item in *propertylist*.

If *propertylist* contains an invalid item, Describe returns an exclamation point (!) for that item and ignores the rest of *propertylist*. Describe returns a question mark (?) if there is no value for a property.

Usage

Specifying the values for *propertylist* can be complex. For information and examples, see the Describe method for the DataWindow control.

Examples

This expression for a computed field in the header band of a DataWindow object displays the DataWindow object's SELECT statement:

```
Describe("DataWindow.Table.Select")
```

See also

Describe on page 451

Exp

Description

Raises *e* to the specified power.

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*.

Examples

This expression returns 7.38905609893065:

```
Exp(2)
```

See also

Log
LogTen
Exp in the *PowerScript Reference*

Fact

Description

Gets the factorial of a number.

Syntax

Fact (*n*)

Argument	Description
n	The number for which you want the factorial

Return value

Double. Returns the factorial of *n*.

Examples

This expression returns 24:

```
Fact(4)
```

Both these expressions return 1:

```
Fact(1)
```

```
Fact(0)
```

See also

Fact in the *PowerScript Reference*

Fill

Description

Builds a string of the specified length by repeating the specified characters until the result string is long enough.

Syntax

Fill (*chars*, *n*)

Argument	Description
chars	A string whose value will be repeated to fill the return string
n	A long whose value is the number of characters in the string you want returned

Return value

String. Returns a string *n* characters long filled with repetitions of 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.

Usage

Fill is used to create a line or other special effect. For example, asterisks repeated in a printed report can fill an amount line, or hyphens can simulate a total line in a screen display.

Examples

This expression returns a string containing 35 stars:

```
Fill("*", 35)
```

This expression returns the string -+-+-+-:

```
Fill("-+", 7)
```

This expression returns 10 tildes (~):

```
Fill("~", 10)
```

See also

Space
Fill in the *PowerScript Reference*

First

Description

Reports the value in the first row in the specified column.

Syntax

First (*column* { FOR *range* { DISTINCT { *expresn* {, *expres2* {, ... } } } } })

Argument	Description
column	The column for which you want the value of the first row. *Column* can be a column name or a column number preceded by a pound sign (#). *Column* can also be an expression that includes a reference to the column.
FOR *range* (optional)	The data that will be included when the value in the first row is found. For most presentation styles, values for *range* are: • ALL — (Default) The value in the first of all rows in *column*. • GROUP *n* — The value in the first of rows in *column* in the specified group. Specify the keyword GROUP followed by the group number: for example, GROUP 1. • PAGE — The value in the first of the rows in *column* on a page. For Graph objects, specify one of the following: • GRAPH — The value in the first row in *column* in the range specified for the Rows option.
DISTINCT (optional)	Causes First to consider only the distinct values in *column* when determining the first value. For a value of *column*, the first row found with the value is used and other rows that have the same value are ignored.
expresn (optional)	One or more expressions that you want to evaluate to determine distinct rows. *Expresn* can be the name of a column, a function, or an expression.

Return value

The datatype of the column. Returns the value in the first row of *column*. If you specify *range*, First returns the value of the first row in *column* in *range*.

Usage

If you specify *range*, First determines the value of the first row in *column* in *range*. If you specify DISTINCT, First returns the first distinct value in *column*, or if you specify *expresn*, the first distinct value in *column* where the value of *expresn* is distinct.

For graphs, you do not select the range when you call the function. The range has already been determined by the Rows setting on the Data property page (the Range property), and the aggregation function uses that range. Settings for Rows include the following:

- For the Graph presentation style, Rows is always All.
- For Graph controls, Rows can be All, Page, or Group.

Not in validation rules or filter expressions
You cannot use this or other aggregate functions in validation rules or filter expressions.

Using an aggregate function cancels the effect of setting Retrieve Rows As Needed in the painter. To do the aggregation, a DataWindow object or a report always retrieves all rows.

Examples

This expression returns the first value in column 3 on the page:

```
First(#3 for page)
```

This expression returns the first distinct value in the column named dept_id in group 2:

```
First(dept_id for group 2 DISTINCT)
```

This expression returns the first value in the column named dept_id in group 2:

```
First(dept_id for group 2)
```

See also

Last

GetRow

Description

Reports the number of a row associated with a band in a DataWindow object or a report.

Syntax

GetRow ( )

Return value

Long. Returns the number of a row if it succeeds, 0 if no data has been retrieved or added, and -1 if an error occurs. Where you call GetRow determines what row it returns, as follows:

If the control in the DataWindow object or report is in this band	GetRow returns
Header	First row on the page
Group header	First row in the group
Detail	The row in which the expression occurs
Group trailer	Last row in the group
Summary	Last row in the report or DataWindow object
Footer	Last row on the page

Examples

This expression for a computed field in the detail band displays the number of each row:

```
GetRow()
```

This expression for a computed field in the header band checks to see if there is data. It returns the number of the first row on the page if there is data, and otherwise returns No Data:

```
If(GetRow()= 0, "No Data", String(GetRow()))
```

See also

"Example 3: creating a row indicator" on page 22
CurrentRow
GetRow on page 505

GetText

Description

Obtains the text that a user has entered in a column.

Syntax

GetText ( )

Return value

String. Returns the text the user has entered in the current column.

Usage

Use GetText in validation rules to compare what the user has entered to application-defined criteria before it is accepted into the data buffer.

Examples

This validation rule checks that the value the user entered in the column is less than 100:

```
Integer(GetText()) < 100
```

See also

GetText on page 512

Hour

Description

Obtains the hour in a time value. The hour is based on a 24-hour clock.

Syntax

Hour (*time*)

Argument	Description
time	The time value from which you want the hour

Return value

Integer. Returns an integer (00–23) containing the hour portion of *time*.

Examples

This expression returns the current hour:

```
Hour(Now())
```

This expression returns 19:

```
Hour(19:01:31)
```

See also

Minute
Now
Second
Hour in the *PowerScript Reference*

If

Description

Evaluates a condition and returns a value based on that condition.

Syntax

If (*boolean*, *truevalue*, *falsevalue*)

Argument	Description
boolean	A boolean expression that evaluates to TRUE or FALSE
truevalue	A string containing the value you want returned if the boolean expression is TRUE
falsevalue	A string containing the value you want returned if the boolean expression is FALSE

Return value

The datatype of *truevalue* or *falsevalue*. Returns *truevalue* if *boolean* is TRUE and *falsevalue* if it is FALSE. Returns NULL if an error occurs.

Examples

This expression returns Boss if salary is over $100,000 and Employee if salary is less than or equal to $100,000:

```
If(salary > 100000, "Boss", "Employee")
```

This expression returns Boss if salary is over $100,000, Supervisor if salary is between $12,000 and $100,000, and Clerk if salary is less than or equal to $12,000:

```
If(salary > 100000, "Boss", If(salary > 12000,
"Supervisor", "Clerk"))
```

In this example of a validation rule, the value the user should enter in the commission column depends on the price. If price is greater than or equal to 1000, then the commission is between .10 and .20. If price is less than 1000, then the commission must be between .04 and .09. The validation rule is:

```
(Number(GetText()) >= If(price >=1000, .10, .04)) AND
(Number(GetText()) <= If(price >= 1000, .20, .09))
```

The accompanying error message expression might be:

```
"Price is " +  If(price >= 1000, "greater than or
equal to", "less than") + " 1000. Commission must be
between " +  If(price >= 1000, ".10", ".04") + " and "
+  If(price >= 1000, ".20.", ".09.")
```

See also

“Example 1: counting NULL values in a column” on page 17
“Example 2: counting male and female employees” on page 18
“Example 3: creating a row indicator” on page 22
“Example 4: displaying all data when a column allows NULLs” on page 23
Case

Int

Description

Gets the largest whole number less than or equal to a number.

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

The datatype of *n*. Returns the largest whole number less than or equal to *n*.

Examples

These expressions return 3.0:

```
Int(3.2)

Int(3.8)
```

These expressions return -4.0:

```
Int(-3.2)

Int(-3.8)
```

See also

Ceiling
Integer
Round
Truncate
Int in the *PowerScript Reference*

Integer

Description

Converts the value of a string to an integer.

Syntax

Integer (*string*)

Argument	Description
string	The string you want returned as an integer

Return value

Integer. Returns the contents of *string* as an integer if it succeeds and 0 if *string* is not a number.

Examples

This expression converts the string 24 to an integer:

```
Integer("24")
```

This expression for a computed field returns "Not a valid age" if age does not contain a number. The expression checks whether the Integer function returns 0, which means it failed to convert the value:

```
If (Integer(age) <> 0, age, "Not a valid age")
```

This expression returns 0:

```
Integer("3ABC")    // 3ABC is not a number
```

This validation rule checks that the value in the column the user entered is less than 100:

```
Integer(GetText()) < 100
```

This validation rule for the column named age insures that age contains a string:

```
Integer(age) <> 0
```

See also

IsNumber
Integer in the *PowerScript Reference*

IsDate

Description

Tests whether a string value is a valid date.

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.

Examples

This expression returns TRUE:

```
IsDate("Jan 1, 99")
```

This expression returns FALSE:

```
IsDate("Jan 32, 1999")
```

This expression for a computed field returns a day number or 0. If the date_received column contains a valid date, the expression returns the number of the day in date_received in the computed field , and otherwise returns 0:

```
If(IsDate(String(date_received)),
DayNumber(date_received), 0)
```

See also IsDate in the *PowerScript Reference*

IsNull

Description Reports whether the value of a column or expression is NULL.

Syntax **IsNull** (*any*)

Argument	Description
any	A column 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.

Examples This expression returns TRUE if either a or b is NULL:

```
IsNull(a + b)
```

This expression returns TRUE if the value in the salary column is NULL:

```
IsNull(salary)
```

This expression returns TRUE if the value the user has entered is NULL:

```
IsNull(GetText())
```

See also "Example 1: counting NULL values in a column" on page 17
"Example 4: displaying all data when a column allows NULLs" on page 23
IsNull in the *PowerScript Reference*

IsNumber

Description Reports whether the value of a string is a number.

Syntax **IsNumber** (*string*)

Argument	Description
string	A string whose value you want to test to determine whether it is a valid number

Return value Boolean. Returns TRUE if *string* is a valid number and FALSE if it is not.

Examples

This expression returns TRUE:

```
IsNumber("32.65")
```

This expression returns FALSE:

```
IsNumber("A16")
```

This expression for a computed field returns "Not a valid age" if age does not contain a number:

```
If(IsNumber(age), age, "Not a valid age")
```

To pass this validation rule, Age_nbr must be a number:

```
IsNumber(Age_nbr) = TRUE
```

See also

Integer
IsNumber in the *PowerScript Reference*

IsRowModified

Description

Reports whether the row has been modified.

Syntax

IsRowModified ( )

Return value

Boolean. Returns TRUE if the row has been modified and FALSE if it has not.

Usage

In a DataWindow object, when you use IsRowModified in bands other than the detail band, it reports on a row in the detail band. See GetRow for a table specifying which row is associated with each band for reporting purposes.

Examples

This expression in a computed field in the detail area displays TRUE or FALSE to indicate whether each row has been modified:

```
IsRowModified()
```

This expression defined in the Properties view for the Color property of the computed field displays the text (TRUE) in red if the user has modified any value in the row:

```
If(IsRowModified(), 255, 0)
```

See also

GetRow

IsRowNew

Description

Reports whether the row has been newly inserted.

Syntax **IsRowNew** ( )

Return value Boolean. Returns TRUE if the row is new and FALSE if it was retrieved from the database.

Usage In a DataWindow object, when you call IsRowNew in bands other than the detail band, it reports on a row in the detail band. See GetRow for a table specifying which row is associated with each band for reporting purposes.

Examples This expression defined in the Properties view for the Protect property of a column prevents the user from modifying the column unless the row has been newly inserted:

```
If(IsRowNew(), 0, 1)
```

See also GetRow
GetItemStatus on page 495

IsSelected

Description Determines whether the row is selected. A selected row is highlighted using reverse video.

Syntax **IsSelected** ( )

Return value Boolean. Returns TRUE if the row is selected and FALSE if it is not selected.

Usage When you use IsSelected in bands other than the detail band, it reports on a row in the detail band. See GetRow for a table specifying which row is associated with each band for reporting purposes.

Examples This expression for a computed field in the detail area displays a bitmap if the row is selected:

```
Bitmap(If(IsSelected(), "beach.bmp", ""))
```

This example allows the DataWindow object to display a salary total for all the selected rows. The expression for a computed field in the detail band returns the salary only when the row is selected so that another computed field in the summary band can add up all the selected salaries.

The expression for cf_selected_salary (the computed field in the detail band) is:

```
If(IsSelected(), salary, 0)
```

The expression for the computed field in the summary band is:

```
Sum(cf_selected_salary for all)
```

See also

GetRow
IsSelected

IsTime

Description

Reports whether the value of a string is a valid time value.

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.

Examples

This expression returns TRUE:

```
IsTime("8:00:00 am")
```

This expression returns FALSE:

```
IsTime("25:00")
```

To pass this validation rule, the value in start_time must be a time:

```
IsTime(start_time)
```

See also

IsTime in the *PowerScript Reference*

Large

Description

Finds a large value at a specified ranking in a column (for example, third-largest, fifth-largest) and returns the value of another column or expression based on the result.

Syntax

Large (*returnexp*, *column*, *ntop* { FOR *range* { DISTINCT { *expres1* {, *expres2* {, ... } } } } })

Argument	Description
returnexp	The value you want returned when the large value is found. *Returnexp* includes a reference to a column, but not necessarily the column that is being evaluated for the largest value, so that a value is returned from the same row that contains the large value.

Argument	Description
column	The column that contains the large value you are searching for. *Column* can be a column name or a column number preceded by a pound sign (#). *Column* can also be an expression that includes a reference to the column. The datatype of *column* must be numeric.
ntop	The ranking of the large value in relation to the column's largest value. For example, when *ntop* is 2, Large finds the second-largest value.
FOR *range* (optional)	The data that will be included when the largest value is found. For most presentation styles, values for *range* are: • ALL — (Default) The largest of all values in *column*. • GROUP *n* — The largest of values in *column* in the specified group. Specify the keyword GROUP followed by the group number: for example, GROUP 1. • PAGE — The largest of the values in *column* on a page. For Graph objects, specify one of the following: • GRAPH — The largest of values in *column* in the range specified for the Rows option.
DISTINCT (optional)	Causes Large to consider only the distinct values in *column* when determining the large value. For a value of *column*, the first row found with the value is used and other rows that have the same value are ignored.
expresn (optional)	One or more expressions that you need to evaluate to determine distinct rows. *Expresn* can be the name of a column, a function, or an expression.

Return value

The datatype of *returnexp*. Returns the *ntop*-largest value if it succeeds and -1 if an error occurs.

Usage

If you specify *range*, Large returns the value in *returnexp* when the value in *column* is the *ntop*-largest value in *range*. If you specify DISTINCT, Large returns *returnexp* when the value in *column* is the *ntop*-largest value of the distinct values in *column*, or if you specify *expresn*, the *ntop*-largest for each distinct value of *expresn*.

For graphs, you do not select the range when you call the function. The range has already been determined by the Rows setting on the Data property page (the Range property), and the aggregation function uses that range. Settings for Rows include the following:

- For the Graph presentation style, Rows is always All
- For Graph controls, Rows can be All, Page, or Group

Max may be faster
If you do not need a return value from another column and you want to find the largest value (*ntop* = 1), use Max; it is faster.

Not in validation rules or filter expressions
You cannot use this or other aggregate functions in validation rules or filter expressions.

Using an aggregate function cancels the effect of setting Retrieve Rows As Needed in the painter. To do the aggregation, a DataWindow object or report always retrieves all rows.

Examples

These expressions return the names of the salespersons with the three largest sales (sum_sales is the sum of the sales for each salesperson) in group 2, which might be the salesregion group. Note that sum_sales contains the values being compared, but Large returns a value in the name column:

```
Large(name, sum_sales, 1 for group 2)
Large(name, sum_sales, 2 for group 2)
Large(name, sum_sales, 3 for group 2)
```

This example reports the salesperson with the third-largest sales, considering only the first entry for each person:

```
Large(name, sum_sales, 3 for all DISTINCT sum_sales)
```

See also

Small

Last

Description

Gets the value in the last row in the specified column.

Syntax

Last (*column* { FOR *range* { DISTINCTT { *expres1* {, *expres2* {, ... } } } } })

Argument	Description
column	The column for which you want the value of the last row. *Column* can be a column name or a column number preceded by a pound sign (#). *Column* can also be an expression that includes a reference to the column.

Argument	Description
FOR *range* (optional)	The data that will be included when the value in the last row is found. For most presentation styles, values for *range* are: • ALL — (Default) The value in the last of all rows in *column*. • GROUP *n* — The value in the last row in *column* in the specified group. Specify the keyword GROUP followed by the group number: for example, GROUP 1. • PAGE — The value in the last row in *column* on a page. For Graph objects, specify one of the following: • GRAPH — The value in the last row in *column* in the range specified for the Rows option.
DISTINCT (optional)	Causes Last to consider only the distinct values in *column* when determining the last value. For a value of *column*, the first row found with the value is used and other rows that have the same value are ignored.
expresn (optional)	One or more expressions that you want to evaluate to determine distinct rows. *Expresn* can be the name of a column, a function, or an expression.

Return value

The datatype of the column. Returns the value in the last row of *column*. If you specify *range*, Last returns the value of the last row in *column* in *range*.

Usage

If you specify *range*, Last determines the value of the last row in *column* in *range*. If you specify DISTINCT, Last returns the last distinct value in *column*, or if you specify *expresn*, the last distinct value in *column* where the value of *expresn* is distinct.

For graphs, you do not select the range when you call the function. The range has already been determined by the Rows setting on the Data property page (the Range property), and the aggregation function uses that range. Settings for Rows include the following:

- For the Graph presentation style, Rows is always All.
- For Graph controls, Rows can be All, Page, or Group.

Not in validation rules or filter expressions
You cannot use this or other aggregate functions in validation rules or filter expressions.

Using an aggregate function cancels the effect of setting Retrieve Rows As Needed in the painter. To do the aggregation, a DataWindow object or report always retrieves all rows.

Examples

This expression returns the last distinct value in the column named dept_id in group 2:

```
Last(dept_id for group 2 DISTINCT)
```

This expression returns the last value in the column named emp_id in group 2:

```
Last(emp_id for group 2)
```

See also

First

LastPos

Description

Finds the last position of a target string in a source string.

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:

```
LastPos("BABE RUTH", "RU", 2)
```

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 "pbsyc80.dll  powerbuilder
// shared  sybase  programs  c:

string  sSource = &
  'c:\programs\sybase\shared\powerbuilder\pbsyc80.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.

Syntax

Left (*string*, *n*)

Argument	Description
string	The string containing the characters you want
n	A long specifying the number of characters you want

Return value

String. Returns the leftmost *n* characters in *string* if it succeeds and the empty string ("") if an error occurs.

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 expression returns BABE:

```
Left("BABE RUTH", 4)
```

This expression returns BABE RUTH:

```
Left("BABE RUTH", 40)
```

This expression for a computed field returns the first 40 characters of the text in the column home_address:

```
Left(home_address, 40)
```

See also

Mid
Pos
Right
Left in the *PowerScript Reference*

LeftTrim

Description

Removes spaces from the beginning of a string.

Syntax

LeftTrim (*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.

Examples

This expression returns RUTH:

```
LeftTrim(" RUTH")
```

This expression for a computed field deletes any leading blanks from the value in the column lname and returns the value preceded by the salutation specified in salut_emp:

```
salut_emp + " " +  LeftTrim(lname)
```

See also

RightTrim
Trim
LeftTrim in the *PowerScript Reference*

Len

Description

Reports the length of a string in characters.

Syntax

Len (*string*)

Argument	Description
string	The string for which you want the length

Return value

Long. Returns a long containing the length of *string* in characters if it succeeds and -1 if an error occurs.

Examples

This expression returns 0:

```
Len("")
```

This validation rule tests that the value the user entered is fewer than 20 characters:

```
Len(GetText()) < 20
```

See also

Len in the *PowerScript Reference*

Log

Description

Gets the natural logarithm of a number.

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.

Inverse
The inverse of the Log function is the Exp function.

Examples

This expression returns 2.302585092:

```
Log(10)
```

This expression returns –.693147 ... :

```
Log(0.5)
```

Both these expressions result in an error during execution:

```
Log(0)
Log(-2)
```

See also

Exp
LogTen
Log in the *PowerScript Reference*

LogTen

Description

Gets the base 10 logarithm of a number.

Syntax

LogTen (*n*)

Argument	Description
n	The number for which you want the base 10 logarithm. The value of *n* must not be negative.

Return value

Double. Returns the base 10 logarithm.

Obtaining a number
The expression 10^n is the inverse for LogTen(*n*). To obtain *n* given number (nbr = LogTen(n)), use *n* = 10^nbr.

Examples

This expression returns 1:

```
LogTen(10)
```

The following expressions both return 0:

```
LogTen(1)
LogTen(0)
```

This expression results in an execution error:

```
LogTen(-2)
```

See also

Log
LogTen in the *PowerScript Reference*

Long

Description

Converts the value of a string to a long.

Syntax

Long (*string*)

Argument	Description
string	The string you want returned as a long

Return value

Long. Returns the contents of *string* as a long if it succeeds and 0 if *string* is not a valid number.

Examples

This expression returns 2167899876 as a long:

```
Long("2167899876")
```

See also

Long in the *PowerScript Reference*

LookUpDisplay

Description

Obtains the display value in the code table associated with the data value in the specified column.

Syntax

LookUpDisplay (*column*)

Argument	Description
column	The column for which you want the code table display value

Return value

String. Returns the display value when it succeeds and the empty string ("") if an error occurs.

Usage

If a column has a code table, a buffer stores a value from the data column of the code table, but the user sees a value from the display column. Use LookUpDisplay to get the value the user sees.

Code tables and data values and graphs
When a column that is displayed in a graph has a code table, the graph displays the data values of the code table by default. To display the display values, call this function when you define the graph data.

Examples

This expression returns the display value for the column unit_measure:

```
LookUpDisplay(unit_measure)
```

Assume the column product_type has a code table and you want to use it as a category for a graph. To display the product type descriptions instead of the data values in the categories, enter this expression in the Category option on the Data page in the graph's property sheet:

```
LookUpDisplay(product_type)
```

Lower

Description

Converts all the characters in a string to lowercase.

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.

Examples

This expression returns babe ruth:

```
Lower("Babe Ruth")
```

See also

Upper
Lower in the *PowerScript Reference*

Match

Description

Determines whether a string's value contains a particular pattern of characters.

Syntax

Match (*string*, *textpattern*)

Argument	Description
string	The string in which you want to look for a pattern of characters

Argument	Description
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.

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:

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 square 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:

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:

This pattern	Matches
AB	Any string that contains AB, such as ABA, DEABC, graphAB_one.
B*	Any string that contains 0 or more Bs, such as AC, B, BB, BBB, ABBBC, and so on. Since B* used alone matches any string, you would not use it alone, but notice its use in some the following examples.
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 string consisting of exactly three digits enclosed in parentheses.

Examples

This validation rule checks that the value the user entered begins with an uppercase letter. If the value of the expression is false, the data fails validation:

```
Match(GetText(), "^[A-Z]")
```

See also

Pos
Match in the *PowerScript Reference*

Max

Description

Gets the maximum value in the specified column.

Syntax

Max (*column* { FOR *range* { DISTINCT { *expres1* {, *expres2* {, ... } } } } })

Argument	Description
column	The column for which you want the maximum value. *Column* can be the column name or the column number preceded by a pound sign (#). *Column* can also be an expression that includes a reference to the column. The datatype of *column* must be numeric.
FOR *range* (optional)	The data that will be included when the maximum value is found. For most presentation styles, values for *range* are: • ALL — (Default) The maximum value of all rows in *column*. • GROUP *n* — The maximum value of rows in *column* in the specified group. Specify the keyword GROUP followed by the group number: for example, GROUP 1. • PAGE — The maximum value of the rows in *column* on a page. For Graph objects, specify one of the following: • GRAPH — The maximum value in *column* in the range specified for the Rows option.
DISTINCT (optional)	Causes Max to consider only the distinct values in *column* when determining the largest value. For a value of *column*, the first row found with the value is used and other rows that have the same value are ignored.
expresn (optional)	One or more expressions that you want to evaluate to determine distinct rows. *Expresn* can be the name of a column, a function, or an expression.

Return value

The datatype of the column. Returns the maximum value in the rows of *column*. If you specify *range*, Max returns the maximum value in *column* in *range*.

Usage

If you specify *range*, Max determines the maximum value in *column* in *range*. If you specify DISTINCT, Max returns the maximum distinct value in *column*, or if you specify *expresn*, the maximum distinct value in *column* where the value of *expresn* is distinct.

For graphs, you do not select the range when you call the function. The range has already been determined by the Rows setting on the Data property page (the Range property), and the aggregation function uses that range. Settings for Rows include the following:

- For the Graph presentation style, Rows is always All.
- For Graph controls, Rows can be All, Page, or Group.

NULL values are ignored and are not considered in determining the maximum.

Not in validation rules or filter expressions
You cannot use this or other aggregate functions in validation rules or filter expressions.

Using an aggregate function cancels the effect of setting Retrieve Rows As Needed in the painter. To do the aggregation, a DataWindow object or report always retrieves all rows.

Examples

This expression returns the maximum of the values in the age column on the page:

```
Max(age for page)
```

This expression returns the maximum of the values in column 3 on the page:

```
Max(#3 for page)
```

This expression returns the maximum of the values in the column named age in group 1:

```
Max(age for group 1)
```

Assuming a DataWindow object displays the order number, amount, and line items for each order, this computed field returns the maximum of the order amount for the distinct order numbers:

```
Max(order_amt for all DISTINCT order_nbr)
```

See also

Min
Max in the *PowerScript Reference*

Median

Description

Calculates the median of the values of the column. The median is the middle value in the set of values, for which there is an equal number of values greater and smaller than it.

Syntax

Median (*column* { FOR *range* { DISTINCT { *expres1* {, *expres2* {, ... } } } } })

Argument	Description
column	The column for which you want the median of the data values. *Column* can be the column name or the column number preceded by a pound sign (#). *Column* can also be an expression that includes a reference to the column. The datatype of *column* must be numeric.
FOR *range* (optional)	The data that will be included in the median. For most presentation styles, values for *range* are: • ALL — (Default) The median of all values in *column*. • GROUP *n* — The median of values in *column* in the specified group. Specify the keyword GROUP followed by the group number: for example, GROUP 1. • PAGE — The median of the values in *column* on a page. For Graph objects, specify one of the following: • GRAPH — The median of values in *column* in the range specified for the Rows.
DISTINCT (optional)	Causes Median to consider only the distinct values in *column* when determining the median. For a value of *column*, the first row found with the value is used and other rows that have the same value are ignored.
expresn (optional)	One or more expressions that you want to evaluate to determine distinct rows. *Expresn* can be the name of a column, a function, or an expression.

Return value

The numeric datatype of the column. Returns the median of the values of the rows in *range* if it succeeds and -1 if an error occurs.

Usage

If you specify *range*, Median returns the median value of *column* in *range*. If you specify DISTINCT, Median returns the median value of the distinct values in *column*, or if you specify *expresn*, the median of *column* for each distinct value of *expresn*.

For graphs, you do not select the range when you call the function. The range has already been determined by the Rows setting on the Data property page (the Range property), and the aggregation function uses that range. Settings for Rows include the following:

- For the Graph presentation style, Rows is always All.
- For Graph controls, Rows can be All, Page, or Group.

In calculating the median, NULL values are ignored.

Not in validation rules or filter expressions
You cannot use this or other aggregate functions in validation rules or filter expressions.

Using an aggregate function cancels the effect of setting Retrieve Rows As Needed in the painter. To do the aggregation, a DataWindow object or report always retrieves all rows.

Examples

This expression returns the median of the values in the column named salary:

```
Median(salary)
```

This expression returns the median of the values in the column named salary of group 1:

```
Median(salary for group 1)
```

This expression returns the median of the values in column 5 on the current page:

```
Median(#5 for page)
```

This computed field returns Above Median if the median salary for the page is greater than the median for the report:

```
If(Median(salary for page) > Median(salary), "Above
Median", " ")
```

This expression for a graph value sets the data value to the median value of the sale_price column:

```
Median(sale_price)
```

This expression for a graph value entered on the Data page in the graph's property sheet sets the data value to the median value of the sale_price column for the entire graph:

```
Median(sale_price for graph)
```

Assuming a DataWindow object displays the order number, amount, and line items for each order, this computed field returns the median of the order amount for the distinct order numbers:

```
Median(order_amt for all DISTINCT order_nbr)
```

See also

Avg
Mode

Mid

Description

Obtains a specified number of characters from a specified position in a string.

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.

Examples

This expression returns "":

```
Mid("BABE RUTH", 40, 5)
```

This expression returns BE RUTH:

```
Mid("BABE RUTH", 3)
```

This expression in a computed field returns ACCESS DENIED if the fourth character in the column password is not R:

```
If(Mid(password, 4, 1) = "R", "ENTER", "ACCESS DENIED")
```

To pass this validation rule, the fourth character in the column password must be 6:

```
Mid(password, 4, 1) = "6"
```

See also

Mid in the *PowerScript Reference*

Min

Description

Gets the minimum value in the specified column.

Syntax

Min (*column* { FOR *range* { DISTINCT { *expres1* {, *expres2* {, ... } } } } })

Argument	Description
column	The column for which you want the minimum value. *Column* can be the column name or the column number preceded by a pound sign (#). *Column* can also be an expression that includes a reference to the column. The datatype of *column* must be numeric.
FOR *range* (optional)	The data that will be included in the minimum. For most presentation styles, values for *range* are: • ALL — (Default) The minimum of all values in *column*. • GROUP *n* — The minimum of values in *column* in the specified group. Specify the keyword GROUP followed by the group number: for example, GROUP 1. • PAGE — The minimum of the values in *column* on a page. For Graph objects, specify one of the following: • GRAPH — The minimum of values in *column* in the range specified for the Rows option.
DISTINCT (optional)	Causes Min to consider only the distinct values in *column* when determining the minimum value. For a value of *column*, the first row found with the value is used and other rows that have the same value are ignored.
expresn (optional)	One or more expressions that you want to evaluate to determine distinct rows. *Expresn* can be the name of a column, a function, or an expression.

Return value

The datatype of the column. Returns the minimum value in the rows of *column*. If you specify *range*, Min returns the minimum value in the rows of *column* in *range*.

Usage

If you specify *range*, Min determines the minimum value in *column* in *range*. If you specify DISTINCT, Min returns the minimum distinct value in *column*, or if you specify *expresn*, the minimum distinct value in *column* where the value of *expresn* is distinct.

For graphs, you do not select the range when you call the function. The range has already been determined by the Rows setting on the Data property page (the Range property), and the aggregation function uses that range. Settings for Rows include:

- For the Graph presentation style, Rows is always All.
- For Graph controls, Rows can be All, Page, or Group.

NULL values are ignored and are not considered in determining the minimum.

Not in validation rules or filter expressions
You cannot use this or other aggregate functions in validation rules or filter expressions.

Using an aggregate function cancels the effect of setting Retrieve Rows As Needed in the painter. To do the aggregation, a DataWindow object or report always retrieves all rows.

Examples

This expression returns the minimum value in the column named age in group 2:

```
Min(age for group 2)
```

This expression returns the minimum of the values in column 3 on the page:

```
Min(#3 for page)
```

Assuming a DataWindow object displays the order number, amount, and line items for each order, this computed field returns the minimum of the order amount for the distinct order numbers:

```
Min(order_amt for all DISTINCT order_nbr)
```

See also

Max
Min in the *PowerScript Reference*

Minute

Description

Obtains the number of minutes in the minutes portion of a time value.

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).

Examples

This expression returns 1:

```
Minute(19:01:31)
```

See also

Hour
Second
Minute in the *PowerScript Reference*

Mod

Description

Obtains the remainder (modulus) of a division operation.

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.

Examples

This expression returns 2:

```
Mod(20, 6)
```

This expression returns 1.5:

```
Mod(25.5, 4)
```

This expression returns 2.5:

```
Mod(25, 4.5)
```

See also

Mod in the *PowerScript Reference*

Mode

Description

Calculates the mode of the values of the column. The mode is the most frequently occurring value.

Syntax

Mode (*column* { FOR *range* { DISTINCT { *expres1* {, *expres2* {, ... } } } } })

Argument	Description
column	The column for which you want the mode of the data values. *Column* can be the column name or the column number preceded by a pound sign (#). *Column* can also be an expression that includes a reference to the column. The datatype of *column* must be numeric.

Argument	Description
FOR *range* (optional)	The data that will be included in the mode. For most presentation styles, values for *range* are: • ALL — (Default) The mode of all values in *column*. • GROUP *n* — The mode of values in *column* in the specified group. Specify the keyword GROUP followed by the group number: for example, GROUP 1. • PAGE — The mode of the values in *column* on a page. For Graph objects, specify one of the following: • GRAPH — The mode of values in *column* in the range specified for the Rows option.
DISTINCT (optional)	Causes Mode to consider only the distinct values in *column* when determining the mode. For a value of *column*, the first row found with the value is used and other rows that have the same value are ignored.
expresn (optional)	One or more expressions that you want to evaluate to determine distinct rows. *Expresn* can be the name of a column, a function, or an expression.

Return value

The numeric datatype of the column. Returns the mode of the values of the rows in *range* if it succeeds and -1 if an error occurs.

Usage

If you specify *range*, Mode returns the mode of *column* in *range*. If you specify DISTINCT, Mode returns the mode of the distinct values in *column*, or if you specify *expresn*, the mode of *column* for each distinct value of *expresn*.

For graphs, you do not select the range when you call the function. The range has already been determined by the Rows setting on the Data property page (the Range property), and the aggregation function uses that range. Settings for Rows include:

- For the Graph presentation style, Rows is always All.
- For Graph controls, Rows can be All, Page, or Group.

In calculating the mode, NULL values are ignored.

Not in validation rules or filter expressions
You cannot use this or other aggregate functions in validation rules or filter expressions.

Using an aggregate function cancels the effect of setting Retrieve Rows As Needed in the painter. To do the aggregation, a DataWindow object or report always retrieves all rows.

Examples

This expression returns the mode of the values in the column named salary:

```
Mode(salary)
```

This expression returns the mode of the values for group 1 in the column named salary:

```
Mode(salary for group 1)
```

This expression returns the mode of the values in column 5 on the current page:

```
Mode(#5 for page)
```

This computed field returns Above Mode if the mode of the salary for the page is greater than the mode for the report:

```
If(Mode(salary for page) > Mode(salary), "Above
Mode", " ")
```

This expression for a graph value sets the data value to the mode of the sale_price column:

```
Mode(sale_price)
```

This expression for a graph value entered on the Data page in the graph's property sheet sets the data value to the mode of the sale_price column for the entire graph:

```
Mode(sale_price for graph)
```

Assuming a DataWindow object displays the order number, amount, and line items for each order, this computed field returns the mode of the order amount for the distinct order numbers:

```
Mode(order_amt for all DISTINCT order_nbr)
```

See also

Avg
Median

Month

Description

Gets the month of a date value.

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*.

Examples

This expression returns 1:

```
Month(1999-01-31)
```

This expression for a computed column returns Wrong Month if the month in the column expected_grad_date is not 6:

```
If(Month(expected_grad_date) = 6, "June", "Wrong
Month")
```

This validation rule expression checks that the value of the month in the date in the column expected_grad_date is 6:

```
Month(expected_grad_date) = 6
```

See also

Day
Date
Year
Month in the *PowerScript Reference*

Now

Description

Obtains the current time based on the system time of the client machine.

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. The timer interval specified for the DataWindow object or report determines the frequency at which the value of Now is updated. For example, if the timer interval is 1 second, it is updated every second.

Examples

This expression returns the current system time:

```
Now()
```

This expression sets the column value to 8:00 when the current system time is before 8:00 and to the current time if it is after 8:00:

```
If(Now() < 08:00:00, '08:00:00', String(Now()))
```

See also

If
Year
Now in the *PowerScript Reference*

Number

Description: Converts a string to a number.

Syntax: **Number** (*string*)

Argument	Description
string	The string you want returned as a number

Return value: A numeric datatype. Returns the contents of *string* as a number. If *string* is not a valid number, Number returns 0.

Examples: This expression converts the string 24 to a number:

```
Number("24")
```

This expression for a computed field tests whether the value in the age column is greater than 55 and if so displays N/A; otherwise, it displays the value in age:

```
If(Number(age) > 55, "N/A", age)
```

This validation rule checks that the number the user entered is between 25,000 and 50,000:

```
Number(GetText())>25000 AND Number (GetText())<50000
```

Page

Description: Gets the number of the current page.

Syntax: **Page** ( )

Return value: Long. Returns the number of the current page.

Calculating the page count
The vertical size of the paper less the top and bottom margins is used to calculate the page count. When the print orientation is landscape, the vertical size of the paper is the shorter dimension of the paper.

Examples: This expression returns the number of the current page:

```
Page()
```

In the DataWindow object or report's footer band, this expression for a computed field displays a string showing the current page number and the total number of pages in the report. The result has the format Page *n* of *total*:

```
'Page ' +  Page() + ' of ' + PageCount()
```

See also

PageAcross
PageCount
PageCountAcross

PageAcross

Description

Gets the number of the current horizontal page. For example, if a report is twice the width of the print preview window and the window is scrolled horizontally to display the portion of the report that was outside the preview, PageAcross returns 2 because the current page is the second horizontal page.

Syntax

PageAcross ( )

Return value

Long. Returns the number of the current horizontal page if it succeeds and -1 if an error occurs.

Examples

This expression returns the number of the current horizontal page:

```
PageAcross()
```

See also

Page
PageCount
PageCountAcross

PageCount

Description

Gets the total number of pages when viewing a DataWindow object or report in Print Preview. This number is also the number of printed pages if the DataWindow object or report is not wider than the preview window. If the DataWindow object or report is wider than the preview window, the number of printed pages will be greater than PageCount gets.

Syntax

PageCount ( )

Return value

Long. Returns the total number of pages.

Usage

PageCount applies to Print Preview.

Calculating the page count
The vertical size of the paper less the top and bottom margins is used to calculate the page count. When the print orientation is landscape, the vertical size of the paper is the shorter dimension of the paper.

Examples

This expression returns the number of pages:

```
PageCount()
```

In the DataWindow object or report's footer band, this expression for a computed field displays a string showing the current page number and the total number of pages in the report. The result has the format *Page n of total*:

```
'Page ' + Page() + ' of ' +  PageCount()
```

See also

Page
PageAcross
PageCountAcross

PageCountAcross

Description

Gets the total number of horizontal pages that are wider than the Print Preview window when a DataWindow object or report is viewed in Print preview.

Syntax

PageCountAcross ( )

Return value

Long. Returns the total number of horizontal pages if it succeeds and -1 if an error occurs.

Usage

PageCountAcross applies to Print Preview.

Examples

This expression returns the number of horizontal pages in the Print Preview window:

```
PageCountAcross()
```

See also

Page
PageAcross
PageCount

Percent

Description

Gets the percentage that the current value represents of the total of the values in the column.

Syntax

Percent (*column* { FOR *range* { DISTINCT { *expres1* {, *expres2* {, ... } } } } })

Argument	Description
column	The column for which you want the value of each row expressed as a percentage of the total of the values of the column. *Column* can be the column name or the column number preceded by a pound sign (#). *Column* can also be an expression that includes a reference to the column. The datatype of *column* must be numeric.
FOR *range* (optional)	The data to be included in the percentage. For most presentation styles, values for *range* are: • ALL — (Default) The percentage that the current value represents of all rows in *column*. • GROUP *n* — The percentage that the current value represents of rows in *column* in the specified group. Specify the keyword GROUP followed by the group number: for example, GROUP 1. • PAGE — The percentage that the current value represents of the rows in *column* on a page. For Graph objects, specify one of the following: • GRAPH — The percentage that the current value represents of values in *column* in the range specified for the Rows option.
DISTINCT (optional)	Causes Percent to consider only the distinct values in *column* when determining the percentage. For a value of *column*, the first row found with the value is used and other rows that have the same value are ignored.
expresn (optional)	One or more expressions that you want to evaluate to determine distinct rows. *Expresn* can be the name of a column, a function, or an expression.

Return value

A numeric datatype (decimal, double, integer, long, or real). Returns the percentage the current row of *column* represents of the total value of the column.

Usage

Usually you use Percent in a column to display the percentage for each row. You can also use Percent in a header or trailer for a group. In the header, Percent displays the percentage for the first value in the group, and in the trailer, for the last value in the group.

If you specify *range*, Percent returns the percentage that the current row of *column* represents relative to the total value of *range*. For example, if column 5 is salary, Percent(#5 for group 1) is equivalent to salary/(Sum(Salary for group 1).

If you specify DISTINCT, Percent returns the percent that a distinct value in *column* represents of the total value of *column*. If you specify *expresn*, Percent returns the percent that the value in *column* represents of the total for *column* in a row in which the value of *expresn* is distinct.

Formatting the percent value
The percentage is displayed as a decimal value unless you specify a format for the result. A display format can be part of the computed field's definition.

For graphs, you do not select the range when you call the function. The range has already been determined by the Rows setting on the Data property page (the Range property), and the aggregation function uses that range. Settings for Rows include the following:

- For the Graph presentation style, Rows is always All.
- For Graph controls, Rows can be All, Page, or Group.

NULL values are ignored and are not considered in the calculation.

Not in validation rules or filter expressions
You cannot use Percent or other aggregate functions in validation rules or filter expressions.

Using an aggregate function cancels the effect of setting Retrieve Rows As Needed in the painter. To do the aggregation, a DataWindow object or report always retrieves all rows.

Examples

This expression returns the value of each row in the column named salary as a percentage of the total of salary:

```
Percent(salary)
```

This expression returns the value of each row in the column named cost as a percentage of the total of cost in group 2:

```
Percent(cost for group 2)
```

This expression entered in the Value box on the Data tab page in the Graph Object property sheet returns the value of each row in the qty_ordered as a percentage of the total for the column in the graph:

```
Percent(qty_ordered for graph)
```

Assuming a DataWindow object displays the order number, amount, and line items for each order, this computed field returns the order amount as a percentage of the total order amount for the distinct order numbers:

```
Percent(order_amt for all DISTINCT order_nbr)
```

See also

CumulativePercent

Pi

Description

Multiplies pi by a specified number.

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.

Usage

Use Pi to convert angles to and from radians.

Examples

This expression returns pi:

```
Pi(1)
```

Both these expressions return the area of a circle with the radius Rad:

```
Pi(1) * Rad^2
Pi(Rad^2)
```

This expression computes the cosine of a 45-degree angle:

```
Cos(45.0 * (Pi(2)/360))
```

See also

Cos
Sin
Tan
Pi in the *PowerScript Reference*

Pos

Description

Finds one string within another string.

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 *string*. 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.

Usage

The Pos function is case sensitive.

Examples

This expression returns the position of the letter *a* in the value of the last_name column:

```
Pos(last_name, "a")
```

This expression returns 6:

```
Pos("BABE RUTH", "RU")
```

This expression returns 1:

```
Pos("BABE RUTH", "B")
```

This expression returns 0 (because the case does not match):

```
Pos("BABE RUTH", "be")
```

This expression returns 0 (because it starts searching at position 5, after the occurrence of BE):

```
Pos("BABE RUTH", "BE", 5)
```

See also

LastPos
Left
Mid
Right
Pos in the *PowerScript Reference*

ProfileInt

Description

Obtains the integer value of a setting in the specified profile file.

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.

Argument	Description
default	An integer value that ProfileInt returns 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*, *key* is not found in *section*, or the value of *key* is not an integer. Returns -1 if an error occurs.

Usage

Use ProfileInt and ProfileString to get configuration settings from a profile file you have designed for your application.

Examples

This example uses the following PROFILE.INI file:

```
[MyApp]
Maximized=1

[Security]
Class = 7
```

This expression tries to return the integer value of the keyword Minimized in section MyApp of file C:\PROFILE.INI. It returns 3 if there is no MyApp section or no Minimized keyword in the MyApp section. Based on the sample file above, it returns 3:

```
ProfileInt("C:\PROFILE.INI", "MyApp", "minimized", 3)
```

See also

ProfileString
ProfileInt in the *PowerScript Reference*

ProfileString

Description

Obtains the string value of a setting in the specified profile file.

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.

Argument	Description
default	A string value that ProfileString returns 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

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 an error occurs, it returns the empty string ("").

Usage

Use ProfileInt and ProfileString to get configuration settings from a profile file you have designed for your application.

Examples

This example uses the following section in PROFILE.INI file:

```
[Employee]
Name="Smith"

[Dept]
Name="Marketing"
```

This expression returns the string for the keyword Name in section Employee in file C:\PROFILE.INI. It returns None if the section or keyword does not exist. In this case it returns Smith:

```
ProfileString("C:\PROFILE.INI", "Employee", "Name",
"None")
```

See also

ProfileInt
ProfileString in the *PowerScript Reference*
SetProfileString in the *PowerScript Reference*

Rand

Description

Obtains a random whole number between 1 and a specified upper limit.

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 cannot exceed 32,767.

Return value

A numeric datatype, the datatype of *n*. Returns a random whole number between 1 and *n*.

Usage

The sequence of numbers generated by repeated calls to the Rand function is a computer-generated pseudorandom sequence.

You can control whether the sequence is different each time your application runs by calling the PowerScript Randomize function to initialize the random number generator.

Examples

This expression returns a random whole number between 1 and 10:

```
Rand(10)
```

See also

Rand in the *PowerScript Reference*
Randomize in the *PowerScript Reference*

Real

Description

Converts a string value to a real datatype.

Syntax

Real (*string*)

Argument	Description
string	The string whose value you want to convert to a real

Return value

Real. Returns the contents of a string as a real. If *string* is not a valid number, Real returns 0.

Examples

This expression converts 24 to a real:

```
Real("24")
```

This expression returns the value in the column temp_text as a real:

```
Real(temp_text)
```

See also

Real in the *PowerScript Reference*

RelativeDate

Description

Obtains the date that occurs a specified number of days after or before another date.

Syntax

RelativeDate (*date*, *n*)

Argument	Description
date	A date value
n	An integer indicating the 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.

Examples

This expression returns 1999-02-10:

```
RelativeDate(1999-01-31, 10)
```

This expression returns 1999-01-21:

```
RelativeDate(1999-01-31, -10)
```

See also

DaysAfter
RelativeDate in the *PowerScript Reference*

RelativeTime

Description

Obtains a time that occurs a specified number of seconds after or before another time within a 24-hour period.

Syntax

RelativeTime (*time*, *n*)

Argument	Description
time	A time value
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.

Examples

This expression returns 19:01:41:

```
RelativeTime(19:01:31, 10)
```

This expression returns 19:01:21:

```
RelativeTime(19:01:31, -10)
```

See also

SecondsAfter
RelativeTime in the *PowerScript Reference*

Replace

Description

Replaces a portion of one string with another.

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 replaces characters in *string1*. The number of characters in *string2* can be greater than, equal to, or fewer 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.

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

This expression changes the last two characters of the string David to e to make it Dave:

```
Replace("David", 4, 2, "e")
```

This expression returns BABY RUTH:

```
Replace("BABE RUTH", 1, 4, "BABY")
```

This expression returns Closed for the Winter:

```
Replace("Closed for Vacation", 12, 8, "the Winter")
```

See also

Replace in the *PowerScript Reference*

RGB

Description

Calculates the long value that represents the color specified by numeric values for the red, green, and blue components of the color.

Syntax

RGB (*red*, *green*, *blue*)

Argument	Description
red	The integer value of the red component of the color
green	The integer value of the green component of the color
blue	The integer value of the blue component of the 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 NULL.

Usage

The formula for combining the colors is:

Red + (256 * Green) + (65536 * Blue)

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.

Determining color components

The value of a component color is an integer between 0 and 255 that represents the amount of the component 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.

The following table lists red, green, and blue values for the 16 standard colors:

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

Color	Red value	Green value	Blue value
Cyan	0	255	255
Dark Cyan	0	128	128
Yellow	255	255	0
Brown	128	128	0

Examples

This expression returns as a long 8421376, which represents dark cyan:

```
RGB(0,128,128)
```

This expression for the Background.Color property of a salary column returns a long that represents red if an employee's salary is greater than $50,000 and white if salary is less than or equal to $50,000:

```
If(salary>50000, RGB(255,0,0), RGB(255,255,255))
```

See also

"Example 3: creating a row indicator" on page 22
RGB in the *PowerScript Reference*

Right

Description

Obtains a specified number of characters from the end of a string.

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 *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 expression returns RUTH:

```
Right("BABE RUTH", 4)
```

This expression returns BABE RUTH:

```
Right("BABE RUTH", 75)
```

See also

Left
Mid
Pos
Right in the *PowerScript Reference*

RightTrim

Description

Removes spaces from the end of a string.

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.

Examples

This expression returns RUTH:

```
RightTrim("RUTH ")
```

See also

LeftTrim
Trim
RightTrim in the *PowerScript Reference*

Round

Description

Rounds a number to the specified number of decimal places.

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*

Return value

Decimal. If *n* is positive, Round returns *x* rounded to the specified number of decimal places. If *n* is negative, it returns *x* rounded to (- *n* +1) places before the decimal point. Returns -1 if it fails.

Examples

This expression returns 9.62:

```
Round(9.624, 2)
```

This expression returns 9.63:

```
Round(9.625, 2)
```

This expression returns 9.600:

```
Round(9.6, 3)
```

This expression returns -9.63:

```
Round(-9.625, 2)
```

This expression returns -10:

```
Round(-9.625, -1)
```

See also

Ceiling
Int
Truncate
Round in the *PowerScript Reference*

RowCount

Description

Obtains the number of rows that are currently available in the primary buffer.

Syntax

RowCount ( )

Return value

Long. Returns the number of rows that are currently available, 0 if no rows are currently available, and -1 if an error occurs.

Examples

This expression in a computed field returns a warning if no data exists and the number of rows if there is data:

```
If(RowCount() = 0, "No Data", String(RowCount()))
```

See also

RowCount on page 570

RowHeight

Description

Reports the height of a row associated with a band in a DataWindow object or a report.

Syntax

RowHeight ( )

Return value

Long. Returns the height of the row in the units specified for the DataWindow object or report if it succeeds, and -1 if an error occurs.

Usage

When you call RowHeight in a band other than the detail band, it reports on a row in the detail band. See GetRow for a table specifying which row is associated with each band for reporting purposes.

Examples

This expression for a computed field in the detail band displays the height of each row:

```
RowHeight()
```

See also

GetRow

Second

Description

Obtains the number of seconds in the seconds portion of a time value.

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).

Examples

This expression returns 31:

```
Second(19:01:31)
```

See also

Hour
Minute
Second in the *PowerScript Reference*

SecondsAfter

Description

Gets the number of seconds one time occurs after another.

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.

Examples

This expression returns 15:

```
SecondsAfter(21:15:30, 21:15:45)
```

This expression returns -15:

```
SecondsAfter(21:15:45, 21:15:30)
```

This expression returns 0:

```
SecondsAfter(21:15:45, 21:15:45)
```

See also

DaysAfter
SecondsAfter in the *PowerScript Reference*

Sign

Description

Reports whether the number is negative, zero, or positive by checking its sign.

Syntax

Sign (*n*)

Argument	Description
n	The number for which you want to determine the sign

Return value

Integer. Returns a number (-1, 0, or 1) indicating the sign of *n*.

Examples

This expression returns 1 (the number is positive):

```
Sign(5)
```

This expression returns 0:

```
Sign(0)
```

This expression returns -1 (the number is negative):

```
Sign(-5)
```

See also

Sign in the *PowerScript Reference*

Sin

Description

Calculates the sine of an angle.

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 it succeeds and -1 if an error occurs.

Examples

This expression returns .8414709848078965:

```
Sin(1)
```

This expression returns 0:

```
Sin(0)
```

This expression returns 0:

```
Sin(pi(1))
```

See also

Cos
Pi
Tan
Sin in the *PowerScript Reference*

Small

Description

Finds a small value at a specified ranking in a column (for example, third-smallest, fifth-smallest) and returns the value of another column or expression based on the result.

Syntax

Small (*returnexp*, *column*, *nbottom* { FOR *range* { DISTINCT { *expres1* {, *expres2* {, ... } } } } })

Argument	Description
returnexp	The value you want returned when the small value is found. *Returnexp* includes a reference to a column, but not necessarily the column that is being evaluated for the small value, so that a value is returned from the same row that contains the small value.
column	The column that contains the small value you are searching for. *Column* can be a column name or a column number preceded by a pound sign (#). *Column* can also be an expression that includes a reference to the column. The datatype of *column* must be numeric.
nbottom	The relationship of the small value to the column's smallest value. For example, when *nbottom* is 2, Small finds the second-smallest value.

Argument	Description
FOR *range* (optional)	The data that will be included when finding the small value. For most presentation styles, values for *range* are: • ALL — (Default) The small value of all rows in *column*. • GROUP *n* — The small value of rows in *column* in the specified group. Specify the keyword GROUP followed by the group number: for example, GROUP 1. • PAGE — The small value of the rows in *column* on a page. For Crosstabs, specify CROSSTAB for *range*: • CROSSTAB — (Crosstabs only) The small value of all rows in *column* in the crosstab. For Graph and OLE objects, specify one of the following: • GRAPH — (Graphs only) The small value in *column* in the range specified for the Rows option. • OBJECT — (OLE objects only) The small value in *column* in the range specified for the Rows option.
DISTINCT (optional)	Causes Small to consider only the distinct values in *column* when determining the small value. For a value of *column*, the first row found with the value is used and other rows that have the same value are ignored.
expresn (optional)	One or more expressions that you want to evaluate to determine distinct rows. *Expresn* can be the name of a column, a function, or an expression.

Return value

The datatype of *returnexp*. Returns the *nbottom*-smallest value if it succeeds and -1 if an error occurs.

Usage

If you specify *range*, Small returns the value in *returnexp* when the value in *column* is the *nbottom*-smallest value in *range*. If you specify DISTINCT, Small returns *returnexp* when the value in *column* is the *nbottom*-smallest value of the distinct values in *column*, or if you specify *expresn*, the *nbottom*-smallest for each distinct value of *expresn*.

For graphs and OLE objects, you do not select the range when you call the function. The range has already been determined by the Rows setting on the Data property page (the Range property), and the aggregation function uses that range. Settings for Rows include the following:

- For the Graph or OLE presentation style, Rows is always All.
- For Graph controls, Rows can be All, Page, or Group.
- For OLE controls, Rows can be All, Current Row, Page, or Group. The available choices depend on the layer the control occupies.

Min may be faster If you do not need a return value from another column and you want to find the smallest value (*nbottom* = 1), use Min; it is faster.

Not in validation rules or filter expressions You cannot use this or other aggregate functions in validation rules or filter expressions.

Using an aggregate function cancels the effect of setting Retrieve Rows As Needed in the painter. To do the aggregation, a DataWindow object or report always retrieves all rows.

Examples

These expressions return the names of the salespersons with the three smallest sales (sum_sales is the sum of the sales for each salesperson) in group 2, which might be the salesregion group. Note that sum_sales contains the values being compared, but Small returns a value in the name column:

```
Small(name, sum_sales, 1 for group 2)
Small(name, sum_sales, 2 for group 2)
Small(name, sum_sales, 3 for group 2)
```

This example reports the salesperson with the third-smallest sales, considering only the first entry for each salesperson:

```
Small(name, sum_sales, 3 for all DISTINCT sum_sales)
```

See also

Large

Space

Description

Builds a string of the specified length whose value consists of spaces.

Syntax

Space (*n*)

Argument	Description
n	A long whose value is the length of the string you want filled with spaces

Return value

String. Returns a string filled with *n* spaces if it succeeds and the empty string ("") if an error occurs.

Examples

This expression for a computed field returns 10 spaces in the computed field if the value of the rating column is Top Secret; otherwise, it returns the value in rating:

```
If(rating = "Top Secret", Space(10), rating)
```

See also

Fill
Space in the *PowerScript Reference*

Sqrt

Description

Calculates the square root of a number.

Syntax

Sqrt (*n*)

Argument	Description
n	The number for which you want the square root

Return value

Double. Returns the square root of *n*.

Usage

Sqrt(*n*) is the same as *n* ^.5.

Taking the square root of a negative number causes an execution error.

Examples

This expression returns 1.414213562373095:

```
Sqrt(2)
```

This expression results in an error at execution time:

```
Sqrt(-2)
```

See also

Sqrt in the *PowerScript Reference*

StDev

Description

Calculates an estimate of the standard deviation for the specified column. Standard deviation is a measurement of how widely values vary from average.

Syntax

StDev (*column* { FOR *range* { DISTINCT { *expres1* {, *expres2* {, ... } } } } })

Argument	Description
column	The column for which you want an estimate for the standard deviation of the values in the rows. *Column* can be the column name or the column number preceded by a pound sign (#). *Column* can also be an expression that includes a reference to the column. The datatype of *column* must be numeric.

Argument	Description
FOR *range* (optional)	The data to be included in the estimate of the standard deviation. For most presentation styles, values for *range* are: • ALL — (Default) The estimate of the standard deviation for all values in *column*. • GROUP *n* — The estimate of the standard deviation for values in *column* in the specified group. Specify the keyword GROUP followed by the group number: for example, GROUP 1. • PAGE — The estimate of the standard deviation for the values in *column* on a page. For Crosstabs, specify CROSSTAB for *range*: • CROSSTAB — (Crosstabs only) The estimate of the standard deviation for all values in *column* in the crosstab. For Graph and OLE objects, specify one of the following: • GRAPH — (Graphs only) The estimate of the standard deviation for values in *column* in the range specified for the Rows option. • OBJECT — (OLE objects only) The estimate of the standard deviation for values in *column* in the range specified for the Rows option.
DISTINCT (optional)	Causes StDev to consider only the distinct values in *column* when determining the standard deviation. For a value of *column*, the first row found with the value is used and other rows that have the same value are ignored.
expresn (optional)	One or more expressions that you want to evaluate to determine distinct rows. *Expresn* can be the name of a column, a function, or an expression.

Return value

Double. Returns an estimate of the standard deviation for *column*.

Usage

If you specify *range*, StDev returns an estimate for the standard deviation of *column* within *range*. If you specify DISTINCT, StDev returns an estimate of the standard deviation for the distinct values in *column*, or if you specify *expresn*, the estimate of the standard deviation of the rows in *column* where the value of *expresn* is distinct.

For graphs and OLE objects, you do not select the range when you call the function. The range has already been determined by the Rows setting on the Data tab page (the Range property), and the aggregation function uses that range. Settings for Rows include the following:

- For the Graph or OLE presentation style, Rows is always All.
- For Graph controls, Rows can be All, Page, or Group.

- For OLE controls, Rows can be All, Current Row, Page, or Group. The available choices depend on the layer the control occupies.

Estimating or calculating actual standard deviation StDev assumes that the values in *column* are a sample of the values in the rows in the column in the database table. If you selected all the rows in the column in the DataWindow object's SELECT statement, use StDevP to compute the standard deviation of the population.

Not in validation rules or filter expressions You cannot use this or other aggregate functions in validation rules or filter expressions.

Using an aggregate function cancels the effect of setting Retrieve Rows As Needed in the painter. To do the aggregation, a DataWindow object or report always retrieves all rows.

Examples

These examples all assume that the SELECT statement did not retrieve all the rows in the database table. StDev is intended to work with a subset of rows, which is a sample of the full set of data.

This expression returns an estimate for standard deviation of the values in the column named salary:

```
StDev(salary)
```

This expression returns an estimate for standard deviation of the values in the column named salary in group 1:

```
StDev(salary for group 1)
```

This expression returns an estimate for standard deviation of the values in column 4 on the page:

```
StDev(#4 for page)
```

This expression entered in the Value box on the Data tab page in the graph's property sheet returns an estimate for standard deviation of the values in the qty_used column in the graph:

```
StDev(qty_used for graph)
```

This expression for a computed field in a crosstab returns the estimate for standard deviation of the values in the qty_ordered column in the crosstab:

```
StDev(qty_ordered for crosstab)
```

Assuming a DataWindow object displays the order number, amount, and line items for each order, this computed field returns the estimated standard deviation of the order amount for the distinct order numbers:

```
StDev(order_amt for all DISTINCT order_nbr)
```

See also

StDevP
Var

StDevP

Description

Calculates the standard deviation for the specified column. Standard deviation is a measurement of how widely values vary from average.

Syntax

StDevP (*column* { FOR *range* { DISTINCT { *expres1* {, *expres2* {, ... } } } } })

Argument	Description
column	The column for which you want the standard deviation of the values in the rows. *Column* can be the column name or the column number preceded by a pound sign (#). *Column* can also be an expression that includes a reference to the column. The datatype of *column* must be numeric.
FOR *range* (optional)	The data to be included in the standard deviation. For most presentation styles, values for *range* are: • ALL — (Default) The standard deviation for all values in *column*. • GROUP *n* — The standard deviation for values in *column* in the specified group. Specify the keyword GROUP followed by the group number: for example, GROUP 1. • PAGE — The standard deviation for the values in *column* on a page. For Crosstabs, specify CROSSTAB for *range*: • CROSSTAB — (Crosstabs only) The standard deviation for all values in *column* in the crosstab. For Graph and OLE objects, specify one of the following: • GRAPH — (Graphs only) The standard deviation for values in *column* in the range specified for the Rows option. • OBJECT — (OLE objects only) The standard deviation for values in *column* in the range specified for the Rows option.
DISTINCT (optional)	Causes StDevP to consider only the distinct values in *column* when determining the standard deviation. For a value of *column*, the first row found with the value is used and other rows that have the same value are ignored.
expresn (optional)	One or more expressions that you want to evaluate to determine distinct rows. *Expresn* can be the name of a column, a function, or an expression.

Return value

Double. Returns the standard deviation for *column*.

Usage

If you specify *range*, StDevP returns the standard deviation for *column* within *range*. If you specify DISTINCT, StDev returns an estimate of the standard deviation for the distinct values in *column*, or if you specify *expresn*, the estimate of the standard deviation of the rows in *column* where the value of *expresn* is distinct.

For graphs and OLE objects, you do not select the range when you call the function. The range has already been determined by the Rows setting on the Data tab page (the Range property), and the aggregation function uses that range. Settings for Rows include the following:

- For the Graph or OLE presentation style, Rows is always All.
- For Graph controls, Rows can be All, Page, or Group.
- For OLE controls, Rows can be All, Current Row, Page, or Group. The available choices depend on the layer the control occupies.

Estimating or calculating actual standard deviation StDevP assumes that the values in *column* are the values in all the rows in the column in the database table. If you did not select all rows in the column in the SELECT statement, use StDev to compute an estimate of the standard deviation of a sample.

Not in validation rules or filter expressions You cannot use this or other aggregate functions in validation rules or filter expressions.

Using an aggregate function cancels the effect of setting Retrieve Rows As Needed in the painter. To do the aggregation, a DataWindow object or report always retrieves all rows.

Examples

These examples all assume that the SELECT statement retrieved all rows in the database table. StDevP is intended to work with a full set of data, not a subset.

This expression returns the standard deviation of the values in the column named salary:

```
StDevP(salary)
```

This expression returns the standard deviation of the values in group 1 in the column named salary:

```
StDevP(salary for group 1)
```

This expression returns the standard deviation of the values in column 4 on the page:

```
StDevP(#4 for page)
```

This expression entered in the Value box on the Data tab page in the graph's property sheet returns the standard deviation of the values in the qty_ordered column in the graph:

```
StDevP(qty_ordered for graph)
```

This expression for a computed field in a crosstab returns the standard deviation of the values in the qty_ordered column in the crosstab:

```
StDevP(qty_ordered for crosstab)
```

Assuming a DataWindow object displays the order number, amount, and line items for each order, this computed field returns the standard deviation of the order amount for the distinct order numbers:

```
StDevP(order_amt for all DISTINCT order_nbr)
```

See also

StDev
VarP

String

Description

Formats data as a string according to a specified display format mask. You can convert and format date, DateTime, numeric, and time data. You can also apply a display format to a string.

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.
format (optional)	A string of the display masks you want to use to format the data. The masks consist of formatting information specific to the datatype of *data*. If *data* is type string, *format* is required. The format string can consist of more than one mask, depending on the datatype of *data*. Each mask is separated by a semicolon. See Usage for details on each datatype.

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 or *format* is not a valid mask.

Usage

For date, DateTime, numeric, and time data, the system's default format is used 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.

A format can include color specifications.

If the display format does not match the datatype, the attempt to apply the mask produces unpredictable results.

For information on specifying display formats, see the *Users Guide*.

When you use String to format a date and the month is displayed as text (for example, when the display format includes "mmm"), the month is in the language of the deployment files available when the application is run. If you have installed localized files in the development environment or on a user's machine, then on that machine the month in the resulting string will be in the language of the localized files.

Examples

This expression returns Jan 31, 1999:

```
String(1999-01-31, "mmm dd, yyyy")
```

This expression returns Jan 31, 1999 6 hrs and 8 min:

```
String(1999-01-31 06:08:00, 'mmm dd, yyyy, h "hrs
and" m "min"')
```

This expression:

```
String(nbr, "0000;(000);****;empty")
```

returns:

0123 if nbr is 123
(123) if nbr is -123
**** if nbr is 0
empty if nbr is NULL

This expression returns A-B-C:

```
String("ABC", "@-@-@")
```

This expression returns A*B:

```
String("ABC", "@*@")
```

This expression returns ABC:

```
String("ABC", "@@@")
```

This expression returns a space:

```
String("ABC", " ")
```

This expression returns 6 hrs and 8 min:

```
String(06:08:02,'h "hrs and" m "min"')
```

This expression returns 08:06:04 pm:

```
String(20:06:04, "hh:mm:ss am/pm")
```

This expression returns 8:06:04 am:

```
String(08:06:04, "h:mm:ss am/pm")
```

This expression returns 6:11:25.300000:

```
String(6:11:25.300000, "h:mm:ss.ffffff")
```

See also String in the *PowerScript Reference*

Sum

Description Calculates the sum of the values in the specified column.

Syntax **Sum** (*column* { FOR *range* { DISTINCT { *expres1* {, *expres2* {, ... } } } } })

Argument	Description
column	The column for which you want the sum of the data values. *Column* can be the column name or the column number preceded by a pound sign (#). *Column* can also be an expression that includes a reference to the column. The datatype of *column* must be numeric.

Argument	Description
FOR *range* (optional)	The data to be included in the sum. For most presentation styles, values for *range* are: • ALL — (Default) The sum of all values in *column*. • GROUP *n* — The sum of values in *column* in the specified group. Specify the keyword GROUP followed by the group number: for example, GROUP 1. • PAGE — The sum of the values in *column* on a page. For Crosstabs, specify CROSSTAB for *range*: • CROSSTAB — (Crosstabs only) The sum of all values in *column* in the crosstab. For Graph and OLE objects, specify one of the following: • GRAPH — (Graphs only) The sum of values in *column* in the range specified for the Rows option of the graph. • OBJECT — (OLE objects only) The sum of values in *column* in the range specified for the Rows option of the OLE object.
DISTINCT (optional)	Causes Sum to consider only the distinct values in *column* when determining the sum. For a value of *column*, the first row found with the value is used and other rows that have the same value are ignored.
expresn (optional)	One or more expressions that you want to evaluate to determine distinct rows. *Expresn* can be the name of a column, a function, or an expression.

Return value

The appropriate numeric datatype. Returns the sum of the data values in *column*.

Usage

If you specify *range*, Sum returns the sum of the values in *column* within *range*. If you specify DISTINCT, Sum returns the sum of the distinct values in *column*, or if you specify *expresn*, the sum of the values of *column* where the value of *expresn* is distinct.

For graphs and OLE objects, you do not select the range when you call the function. The range has already been determined by the Rows setting on the Data property page (the Range property), and the aggregation function uses that range. Settings for Rows include the following:

- For the Graph or OLE presentation style, Rows is always All.
- For Graph controls, Rows can be All, Page, or Group.
- For OLE controls, Rows can be All, Current Row, Page, or Group. The available choices depend on the layer the control occupies.

NULL values are ignored and are not included in the calculation.

Not in validation rules or filter expressions
You cannot use this or other aggregate functions in validation rules or filter expressions.

Using an aggregate function cancels the effect of setting Retrieve Rows As Needed in the painter. To do the aggregation, a DataWindow object or report always retrieves all rows.

Examples

This expression returns the sum of the values in group 1 in the column named salary:

```
Sum(salary for group 1)
```

This expression returns the sum of the values in column 4 on the page:

```
Sum(#4 for page)
```

Assuming a DataWindow object displays the order number, amount, and line items for each order, this computed field returns the sum of the order amount for the distinct order numbers:

```
Sum(order_amt for all DISTINCT order_nbr)
```

See also

"Example 1: counting NULL values in a column" on page 17
"Example 2: counting male and female employees" on page 18

Tan

Description

Calculates the tangent of an angle.

Syntax

Tan (*n*)

Argument	Description
n	The angle (in radians) for which you want the tangent

Return value

Double. Returns the tangent of *n* if it succeeds and -1 if an error occurs.

Examples

Both these expressions return 0:

```
Tan(0)
Tan(Pi(1))
```

This expression returns 1.55741:

```
Tan(1)
```

See also

Cos
Pi
Sin
Tan in the *PowerScript Reference*

Time

Description

Converts a string to a time datatype.

Syntax

Time (*string*)

Argument	Description
string	A string containing a valid time (such as 8 AM or 10:25) that you want returned as a time datatype. 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 for minutes and seconds is 00 and for microseconds is 000000. AM or PM is determined automatically.

Return value

Time. Returns the time in *string* as a time datatype. If *string* does not contain a valid time, Time returns 00:00:00.

Examples

This expression returns the time datatype for 45 seconds before midnight (23:59:15):

```
Time("23:59:15")
```

This expression for a computed field returns the value in the time_received column as a value of type time if time_received is not the empty string. Otherwise, it returns 00:00:00:

```
If(time_received = "" ,00:00:00,
Time(time_received))
```

This example is similar to the previous one, except that it returns 00:00:00 if time_received contains a NULL value:

```
If(IsNull(time_received), 00:00:00,
Time(time_received))
```

See also

Time in the *PowerScript Reference*

Today

Description: Obtains the system date and time.

Syntax: **Today** ( )

Return value: DateTime. Returns the current system date and time.

Usage: To display both the date and the time, a computed field must have a display format that includes the time.

The PowerScript and DataWindow painter versions of the Today function have different datatypes. The return value of the PowerScript Today function is date.

Examples: This expression for a computed field displays the date and time when the display format for the field is "mm/dd/yy hh:mm":

```
Today()
```

See also: Now
Today in the *PowerScript Reference*

Trim

Description: Removes leading and trailing spaces from a string.

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.

Usage: Trim is useful for removing spaces that a user might have typed before or after newly entered data.

Examples: This expression returns BABE RUTH:

```
Trim(" BABE RUTH ")
```

See also: LeftTrim
RightTrim
Trim in the *PowerScript Reference*

Truncate

Description

Truncates a number to the specified number of decimal places.

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*

Return value

The datatype of *x*. If *n* is positive, returns *x* truncated to the specified number of decimal places. If *n* is negative, returns *x* truncated to (- *n* +1) places before the decimal point. Returns -1 if it fails.

Examples

This expression returns 9.2:

```
Truncate(9.22, 1)
```

This expression returns 9.2:

```
Truncate(9.28, 1)
```

This expression returns 9:

```
Truncate(9.9, 0)
```

This expression returns -9.2:

```
Truncate(-9.29, 1)
```

This expression returns 0:

```
Truncate(9.2, -1)
```

This expression returns 50:

```
Truncate(54, -1)
```

See also

Ceiling
Int
Round
Truncate in the *PowerScript Reference*

Upper

Description

Converts all characters in a string to uppercase letters.

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.

Examples

This expression returns BABE RUTH:

```
Upper("Babe Ruth")
```

See also

Lower
Upper in the *PowerScript Reference*

Var

Description

Calculates an estimate of the variance for the specified column. The variance is the square of the standard deviation.

Syntax

Var (*column* { FOR *range* { DISTINCT { *expres1* {, *expres2* {, ... } } } } })

Argument	Description
column	The column for which you want an estimate for the variance of the values in the rows. *Column* can be the column name or the column number preceded by a pound sign (#). *Column* can also be an expression that includes a reference to the column. The datatype of *column* must be numeric.

Argument	Description
FOR *range* (optional)	The data to be included in the estimate of the variance. For most presentation styles, values for *range* are: • ALL — (Default) The estimate of the variance for all rows in *column*. • GROUP *n* — The estimate of the variance for rows in *column* in the specified group. Specify the keyword GROUP followed by the group number: for example, GROUP 1. • PAGE — The estimate of the variance for the rows in *column* on a page. For Crosstabs, specify CROSSTAB for *range*: • CROSSTAB — (Crosstabs only) The estimate of the variance for all rows in *column* in the crosstab. For Graph and OLE objects, specify one of the following: • GRAPH — (Graphs only) The estimate of the variance for rows in *column* in the range specified for the Rows option. • OBJECT — (OLE objects only) The estimate of the variance for rows in *column* in the range specified for the Rows option.
DISTINCT (optional)	Causes Var to consider only the distinct values in *column* when determining the variance. For a value of *column*, the first row found with the value is used and other rows that have the same value are ignored.
expresn (optional)	One or more expressions that you want to evaluate to determine distinct rows. *Expresn* can be the name of a column, a function, or an expression.

Return value

Double. Returns an estimate for the variance for *column*. If you specify *group*, Var returns an estimate for the variance for *column* within *group*.

Usage

If you specify *range*, Var returns an estimate for the variance for *column* within *range*. If you specify DISTINCT, Var returns the variance for the distinct values in *column*, or if you specify *expresn*, the estimate for the variance of the rows in *column* where the value of *expresn* is distinct.

For graphs and OLE objects, you do not select the range when you call the function. The range has already been determined by the Rows setting on the Data property page (the Range property), and the aggregation function uses that range. Settings for Rows include the following:

- For the Graph or OLE presentation style, Rows is always All.
- For Graph controls, Rows can be All, Page, or Group.

- For OLE controls, Rows can be All, Current Row, Page, or Group. The available choices depend on the layer the control occupies.

Estimating variance or calculating actual variance Var assumes that the values in *column* are a sample of the values in rows in the column in the database table. If you select all rows in the column in the SELECT statement, use VarP to compute the variance of a population.

Not in validation rules or filter expressions You cannot use this or other aggregate functions in validation rules or filter expressions.

Using an aggregate function cancels the effect of setting Retrieve Rows As Needed in the painter. To do the aggregation, a DataWindow object or report always retrieves all rows.

Examples

These examples all assume that the SELECT statement did not retrieve all of the rows in the database table. Var is intended to work with a subset of rows, which is a sample of the full set of data.

This expression returns an estimate for the variance of the values in the column named salary:

```
Var(salary)
```

This expression returns an estimate for the variance of the values in the column named salary in group 1:

```
Var(salary for group 1)
```

This expression entered in the Value box on the Data property page in the graph's property sheet returns an estimate for the variance of the values in the quantity column in the graph:

```
Var(quantity for graph)
```

This expression for a computed field in a crosstab returns an estimate for the variance of the values in the quantity column in the crosstab:

```
Var(quantity for crosstab)
```

Assuming a DataWindow object displays the order number, amount, and line items for each order, this computed field returns the estimate for the variance of the order amount for the distinct order numbers:

```
Var(order_amt for all DISTINCT order_nbr)
```

See also

StDev
VarP

VarP

Description

Calculates the variance for the specified column. The variance is the square of the standard deviation.

Syntax

VarP (*column* { FOR *range* { DISTINCT { *expres1* {, *expres2* {, ... } } } } })

Argument	Description
column	The column for which you want the variance of the values in the rows. *Column* can be the column name or the column number preceded by a pound sign (#). *Column* can also be an expression that includes a reference to the column. The datatype of *column* must be numeric.
FOR *range* (optional)	The data that will be included in the variance. For most presentation styles, values for *range* are: • ALL — (Default) The variance for all rows in *column*. • GROUP *n* — The variance for rows in *column* in the specified group. Specify the keyword GROUP followed by the group number: for example, GROUP 1. • PAGE — The variance for the rows in *column* on a page. For Crosstabs, specify CROSSTAB for *range*: • CROSSTAB — (Crosstabs only) The variance for all rows in *column* in the crosstab. For Graph and OLE objects, specify one of the following: • GRAPH — (Graphs only) The variance for rows in *column* in the range specified for the Rows option. • OBJECT — (OLE objects only) The variance for rows in *column* in the range specified for the Rows option.
DISTINCT (optional)	Causes VarP to consider only the distinct values in *column* when determining the variance. For a value of *column*, the first row found with the value is used and other rows that have the same value are ignored.
expresn (optional)	One or more expressions that you want to evaluate to determine distinct rows. *Expresn* can be the name of a column, a function, or an expression.

Return value

Double. Returns the variance for *column*. If you specify *group*, Var returns the variance for *column* within *range*.

Usage

If you specify *range*, VarP returns the variance for *column* within *range*. If you specify DISTINCT, VarP returns the variance for the distinct values in *column*, or if you specify *expresn*, the variance of the rows in *column* where the value of *expresn* is distinct.

For graphs and OLE objects, you do not select the range when you call the function. The range has already been determined by the Rows setting on the Data property page (the Range property), and the aggregation function uses that range. Settings for Rows include the following:

- For the Graph or OLE presentation style, Rows is always All.
- For Graph controls, Rows can be All, Page, or Group.
- For OLE controls, Rows can be All, Current Row, Page, or Group. The available choices depend on the layer the control occupies.

Estimating variance or calculating actual variance VarP assumes that the values in *column* are the values in all rows in the column in the database table. If you did not select all the rows in the column in the SELECT statement, use Var to compute an estimate of the variance of a sample.

Not in validation rules or filter expressions You cannot use this or other aggregate functions in validation rules or filter expressions.

Using an aggregate function cancels the effect of setting Retrieve Rows As Needed in the painter. To do the aggregation, a DataWindow object or report always retrieves all rows.

Examples

These examples all assume that the SELECT statement retrieved all rows in the database table. VarP is intended to work with a full set of data, not a subset.

This expression returns the variance of the values in the column named salary:

```
VarP(salary)
```

This expression returns the variance of the values in group 1 in the column named salary:

```
VarP(salary for group 1)
```

This expression returns the variance of the values in column 4 on the page:

```
VarP(#4 for page)
```

This expression entered in the Value box on the Data property page in the graph's property sheet returns the variance of the values in the quantity column in the graph:

```
VarP(quantity for graph)
```

This expression for a computed field in a crosstab returns the variance of the values in the quantity column in the crosstab:

```
VarP(quantity for crosstab)
```

Assuming a DataWindow object displays the order number, amount, and line items for each order, this computed field returns the variance of the order amount for the distinct order numbers:

```
VarP(order_amt for all DISTINCT order_nbr)
```

See also

StDevP
Var

WordCap

Description

Sets the first letter of each word in a string to a capital letter and all other letters to lowercase (for example, ROBERT E. LEE would be Robert E. Lee).

Syntax

WordCap (*string*)

Argument	Description
string	A string or expression that evaluates to a string that you want to display with initial capital letters (for example, Monday Morning)

Return value

String. Returns *string* with the first letter of each word set to uppercase and the remaining letters lowercase if it succeeds, and NULL if an error occurs.

Examples

This expression returns Boston, Massachusetts:

```
WordCap("boston, MASSACHUSETTS")
```

This expression concatenates the characters in the emp_fname and emp_lname columns and makes the first letter of each word uppercase:

```
WordCap(emp_fname + " " + emp_lname)
```

Year

Description

Gets the year of a date value.

Syntax

Year (*date*)

Argument	Description
date	The date value 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 the year is two digits, then the century is set as follows. If the year is between 00 to 49, the first two digits are 20; if the year is between 50 and 99, the first two digits are 19.

Usage

Obtains the year portion of *date*. Years from 1000 to 3000 inclusive are handled.

If your data includes dates before 1950, such as birth dates, always specify a 4-digit year so that Year (and other functions, such as Sort) interpret the date as intended.

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 to have the application change it (by calling the RegistrySet function). The user might need to reboot after the setting is changed.

Examples

This expression returns 1999:

```
Year(1999-01-31)
```

See also

Day
Month
Year in the *PowerScript Reference*

CHAPTER 3

DataWindow Object Properties

About this chapter

This chapter describes the properties that control the appearance and behavior of a DataWindow object.

Contents

Overview of DataWindow object properties

There are many ways you can affect the values of DataWindow object properties during execution:

- There are several methods that get and set specific properties, and you can use the general-purpose Describe and Modify functions to get and set property values.
- For many properties, you can enter expressions in the painter that set properties conditionally at execution time.
- You can use the SyntaxFromSQL method on a transaction object to generate DataWindow source code that sets some DataWindow properties. You can use the generated code in the Create function to create new DataWindows.

Summary tables in the first part of this chapter

The tables in “Controls in a DataWindow and their properties” on page 128 list the properties for each control within a DataWindow object, with short descriptions. There are also tables for SyntaxFromSQL object keywords. After the first table of DataWindow properties, the tables are alphabetical by control and keyword name.

The tables include check mark columns that identify whether you can use a property with Modify (*M*) or SyntaxFromSQL (*S*). When (*exp*) is included in the description, you can specify a DataWindow expression as the value for that property. A DataWindow expression lets you specify conditions for determining the property value.

You can get the value of all properties in all tables
During execution, you can use Describe or dot notation to get the value of all properties listed in all tables. The tables do not show checkmarks to account for this capability.

Alphabetical reference list in the second part of this chapter

The second half of this chapter is an alphabetical list of properties with descriptions, syntax, and examples. When you find a property you want to use in the first part, look up the property in the alphabetical list to find the specific syntax you need to use. In the tables that describe the property values, (*exp*) again indicates that you can use a DataWindow expression for the value.

Accessing properties in different DataWindow environments The property reference has syntax for Describe and Modify and for PocketBuilder dot notation.

Examples and quoted strings The arguments for Describe and Modify are quoted strings that are generally valid in all environments. If the strings include nested quotes, see "Nested strings and special characters for DataWindow object properties" on page 352 for information on the appropriate escape character in each environment.

For more information and examples of setting properties, see:

- Chapter 5, "Accessing DataWindow Object Properties in Code"
- Describe and Modify methods in Chapter 9, "Methods for the DataWindow Control"
- SyntaxFromSQL method in the *PowerScript Reference*

Controls in a DataWindow and their properties

The tables in this section list the properties that apply to DataWindow objects and SyntaxFromSQL keywords.

Properties for the DataWindow object

An x in the M (Modify) column means you can change the property. An x in the S column means you can use the property with the SyntaxFromSQL method. When (*exp*) is included in the description, you can specify a DataWindow expression as the value for that property.

Property for the DataWindow	M	S	Description
Attributes			All general properties.
Bands			List of bands.
Bandname.property	x		Color, height, and so on for a band, where *bandname* is Detail, Footer, Header, Summary, or Trailer.
Bandname.Text	x		Rich text content where *bandname* is Detail, Footer, or Header.
Color	x	x	Background color.
Column.Count			Number of columns.
Data			Description of data.
Data.HTML			Description of the data and format of the DataWindow in HTML format.

Property for the DataWindow	M	S	Description
Data.HTMLTable			Description of the data in the DataWindow in HTML table format.
Detail.property	x		Color, height, and so on for the detail band.
FirstRowOnPage			The row number of the first displayed row.
Font.Bias	x		Treat fonts as display or printer.
Footer.property	x		Color, height, and so on for the footer band (see *Bandname.property* in this table).
Grid.ColumnMove	x		Whether the user can drag to reposition columns.
Grid.Lines	x		Options for lines in grid DataWindow and crosstab.
Header.#.property	x		Color, height, and so on for a group's header band.
Header.property	x		Color, height, and so on for the header band.
Help.property	x		Help settings for DataWindow actions.
HideGrayLine	x		Whether a gray line displays at page boundaries.
HorizontalScrollMaximum			Width of scroll box in the horizontal scroll bar.
HorizontalScrollMaximum2			Width of second scroll box when scroll bar is split.
HorizontalScrollPosition	x		Position of the scroll box in the scroll bar.
HorizontalScrollPosition2	x		Position of scroll box in second split scroll bar.
HorizontalScrollSplit	x		The position of the split in the scroll bar.
HTMLDW	x		(*exp*) Whether HTML for the DataWindow is interactive and coordinated with a server component for retrievals and updates.
HTMLGen.property	x		(*exp*) Settings for HTML generation.
HTMLTable.property	x		Settings for the display of DataWindow data when displayed in HTML table format.
Label.property	x	x	Settings for the Label presentation style.
LastRowOnPage			The last visible row on the page.
Message.Title	x	x	The title of the dialog box that displays errors.
NoUserPrompt	x		Determines whether an error message is displayed to the user.
Objects			The controls in the DataWindow.
Pointer	x		(*exp*) The pointer when over the DataWindow.
Print.Buttons	x		Whether buttons display on the printed output.
Print.Preview.Buttons	x		Whether buttons display in print preview.
Print.property	x	x	Various settings for printing.
Printer	x		The currently selected printer.
Processing			Processing required by the presentation style.
QueryMode	x		Whether the DataWindow is in query mode.
QuerySort	x		Whether to sort the result set from the query.
ReadOnly	x		Whether the DataWindow is read-only.
Retrieve.AsNeeded	x		Whether to retrieve data only as needed.
Row.Resize	x		Whether user can change the height of rows.
Selected	x		List of selected controls.

Property for the DataWindow	M	S	Description
Selected.Data			List of selected data.
Selected.Mouse	x		Whether user can use the mouse to select.
ShowDefinition	x		(*exp*) Display column names instead of data.
Sparse	x		(*exp*) The repeating columns to be suppressed.
Storage			The amount of storage used by DataWindow.
StoragePageSize			The default page size for DataWindow storage.
Summary.property	x		Color, height, and so on for the summary band.
Syntax			The syntax of the DataWindow.
Syntax.Data			The data of the DataWindow in parse format.
Syntax.Modified	x		Whether the syntax has been modified.
Table.property	x		Various settings for the database.
Table.sqlaction.property	x		Stored procedures for update activity.
Timer_Interval	x	x	The milliseconds between timer events.
Trailer.#.property	x		Color, height, and so on for a group's trailer band.
Units		x	The unit of measure for the DataWindow.
VerticalScrollMaximum			The height of the scroll box in the scroll bar.
VerticalScrollPosition	x		The position of the scroll box in the scroll bar.
Zoom	x		The scaling percentage of the DataWindow.

Properties for Button controls in DataWindow objects

An x in the M (Modify) column means you can change the property. When (*exp*) is included in the description, you can specify a DataWindow expression as the value for that property.

Property for a Button	M	Description
Action	x	(*exp*) The action a user can assign to the button.
Attributes		A list of the properties of the button control.
Background.property	x	(*exp*) Background settings for the button control.
Band		The band containing the button control.
Color	x	(*exp*) The text color.
DefaultPicture	x	Whether or not the action's default picture is to be used on the button (user-defined action has no default picture).
Enabled	x	Determines whether a button control on a DataWindow is enabled.
Filename	x	(*exp*) Name of the file containing the picture to be used on the button (if not specified, just the text is used).
Font.property	x	(*exp*) Font settings for the text.

Property for a Button	M	Description
HTextAlign	x	(*exp*) How the text in the button is horizontally aligned. Values are: 0 (center), 1 (left), 2 (right).
Height	x	(*exp*) The height of the button control.
HideSnaked	x	Whether the button control appears once per page when printing newspaper columns.
Moveable	x	Whether the user can move the button control.
Name		The name of the button control.
Pointer	x	(*exp*) The pointer image when it is over the button control.
Resizeable	x	Whether the user can resize the button control.
SlideLeft	x	(*exp*) Whether the button control moves left to fill in empty space.
SlideUp	x	(*exp*) How the button control moves up to fill in empty space.
SuppressEventProcessing	x	Whether or not ButtonClicked and ButtonClicking events are fired for this particular button.
Tag	x	(*exp*) The tag text for the button control.
Text	x	(*exp*) The displayed text.
Type		The control's type, which is button.
VTextAlign	x	(*exp*) How the text in the button is vertically aligned. Values are: 0 (center), 1 (top), 2 (bottom), 3 (multiline).
Visible	x	(*exp*) Whether the button control is visible.
Width	x	(*exp*) The width of the button control.
X	x	(*exp*) The x coordinate of the button control.
Y	x	(*exp*) The y coordinate of the button control.

Properties for Column controls in DataWindow objects

An x in the M (Modify) column means you can change the property. An x in the S column means you can use the property with the SyntaxFromSQL method. When (*exp*) is included in the description, you can specify a DataWindow expression as the value for that property.

Property for a Column	M	S	Description
Accelerator	x		(*exp*) The accelerator key for the column
Alignment	x		(*exp*) The alignment of the column's text
Attributes			A list of the properties of the column
Background.property	x	x	(*exp*) Background settings for the column

Property for a Column	M	S	Description
Band			The band containing the column
BitmapName			Whether the column's content names a picture that will be displayed instead of the text
Border	x	x	(*exp*) The type of border around the column
CheckBox.property	x		Settings for CheckBox edit style
Color	x	x	(*exp*) The text color
ColType			The column's datatype
Criteria.property	x		Settings for column in Prompt for Criteria dialog box
dbName	x		The name of the database column
dddw.property	x		Settings for DropDownDataWindow edit style
ddlb.property	x		Settings for DropDownListBox edit style
Edit.property	x	x	Settings for Edit edit style
EditMask.property	x		Settings for EditMask edit style
Font.property	x	x	(*exp*) Font settings for the column text
Format	x		(*exp*) The column's display format
Height	x		(*exp*) The height of the column
Height.AutoSize	x		Whether column height is adjusted to fit the data
HideSnaked	x		Whether the control appears once per page when printing newspaper columns
HTML.property	x		(*exp*) Settings for creating hyperlinks for column data
ID			The number of the column
Identity	x		Whether the DBMS sets the column's value
Initial	x		The initial value in the column for a new row
Key	x		Whether column is part of the table's primary key
LineRemove	x		Whether to remove line of text when the column is not visible
Moveable	x		Whether the user can move the column
Multiline	x		Whether the column can be multiline
Name			The name of the column
Pointer	x		(*exp*) The pointer's image when it is over column
Protect	x		(*exp*) Whether column is protected from changes
RadioButtons.property	x		Settings for RadioButton edit style
Resizeable	x		Whether the user can resize the column
SlideLeft	x		(*exp*) Whether column moves left to fill in space
SlideUp	x		(*exp*) How the column moves up to fill in space
TabSequence	x		The position of the column in the tab order
Tag	x		(*exp*) The tag text for the column

Property for a Column	M	S	Description
Type			The control's type, which is Column
Update	x		Whether the column is updatable
Validation	x		(*exp*) The validation expression for the column
ValidationMsg	x		(*exp*) The message displayed when validation fails
Values (for columns)	x		The values in the column's code table
Visible	x		(*exp*) Whether the column control is visible
Width	x		(*exp*) The width of the column
Width.Autosize	x		Whether width adjusts to the data
X	x		(*exp*) The x coordinate of the column
Y	x		(*exp*) The y coordinate of the column

Properties for Computed Field controls in DataWindow objects

An x in the M (Modify) column means you can change the property. When (*exp*) is included in the description, you can specify a DataWindow expression as the value for that property.

Property for a computed field	M	Description
Alignment	x	(*exp*) The alignment of the computed field's text
Attributes		A list of the properties of the computed field
Background.property	x	(*exp*) Background settings for the computed field
Band		The band containing the computed field
Border	x	(*exp*) The type of border around the computed field
Color	x	(*exp*) The text color
ColType		The column's datatype
Expression	x	The expression for the computed field
Font.property	x	(*exp*) Font settings for the computed field
Format	x	(*exp*) The computed field's display format
Height	x	(*exp*) The height of the computed field
Height.AutoSize	x	Whether the computed field's height is adjusted to fit the data
HideSnaked	x	Whether the control appears once per page when printing newspaper columns
HTML.property	x	(*exp*) Settings for creating hyperlinks for the computed field
LineRemove	x	Whether to remove line of text when the computed field is not visible

Property for a computed field	M	Description
Moveable	x	Whether the user can move the computed field
Multiline	x	Whether the column can be multiline
Name		The name of the computed field
Pointer	x	(*exp*) The pointer image when it is over the computed field
Resizeable	x	Whether the user can resize the computed field
SlideLeft	x	(*exp*) Whether the computed field moves left to fill in space
SlideUp	x	(*exp*) How the computed field moves up to fill in empty space
Tag	x	(*exp*) The tag text for the computed field
Type		The control's type, which is Compute
Visible	x	(*exp*) Whether the computed field control is visible
Width	x	(*exp*) The width of the computed field
Width.Autosize	x	Whether width adjusts to the data
X	x	(*exp*) The x coordinate of the computed field
Y	x	(*exp*) The y coordinate of the computed field

Properties for Graph controls in DataWindow objects

An x in the M (Modify) column means you can change the property. When (*exp*) is included in the description, you can specify a DataWindow expression as the value for that property.

Property for a Graph	M	Description
Attributes		A list of the properties of the graph
Axis	x	(*exp*) List of items (categories, series, or values) for the axis
Axis.property	x	(*exp*) Properties for a graph axis
Axis.DispAttr	x	(*exp*) Display properties for an axis (see DispAttr.*fontproperty* in this table)
BackColor	x	(*exp*) The background color of the graph
Band		The band containing the graph
Border	x	(*exp*) The type of border around the graph
Category	x	(*exp*) List of categories for the axis (see *Axis* in this table)
Category.property	x	(*exp*) Properties for the Category axis (see *Axis.property* in this table)

Property for a Graph	M	Description
Category.DispAttr	x	(*exp*) Display properties for the Category axis (see DispAttr.*fontproperty* in this table)
Color	x	(*exp*) The text color
Depth	x	(*exp*) The depth of a 3D graph
DispAttr.fontproperty	x	Font settings for various components of the graph
Elevation	x	(*exp*) The elevation of a 3D graph
GraphType	x	(*exp*) The type of graph (pie, bar, and so on)
Height	x	(*exp*) The height of the graph
HideSnaked	x	Whether the control appears once per page when printing newspaper columns
Legend	x	(*exp*) The location of the legend
Legend.DispAttr.fontproperty	x	(*exp*) Display properties for the legend
Moveable	x	Whether the user can move the graph
Name		The name of the graph control
OverlapPercent	x	(*exp*) The overlap between data markers in different series
Perspective	x	(*exp*) The distance of the graph from the front of the window
Pie.DispAttr.fontproperty	x	(*exp*) Display properties for the pie slice labels
Pointer	x	(*exp*) The pointer image when it is over the graph
Range		The rows in the DataWindow that are included in the graph
Resizeable	x	Whether the user can resize the graph
Rotation	x	(*exp*) The left-to-right rotation of a 3D graph
Series	x	(*exp*) List of series for the axis (see *Axis* in the table)
Series.property	x	(*exp*) Properties for the Series axis (see *Axis.property* in this table)
Series.DispAttr	x	(*exp*) Display properties for the Series axis (see DispAttr.*fontproperty* in this table)
ShadeColor	x	(*exp*) The color of the back edge for 3D data markers
SizeToDisplay	x	(*exp*) Whether to size the graph to the display area
SlideLeft	x	(*exp*) Whether the graph moves left to fill in empty space
SlideUp	x	(*exp*) How the graph moves up to fill in empty space

Property for a Graph	M	Description
Spacing	x	(*exp*) The gap between categories
Tag	x	(*exp*) The tag text for the graph
Title	x	(*exp*) The graph's title
Title.DispAttr.fontproperty	x	(*exp*) Display properties for the title
Type		The control's type, which is graph
Values	x	(*exp*) List of values for the axis (see *Axis* in the table)
Values.property	x	(*exp*) Properties for the Values axis (see *Axis.property* in the table)
Values.DispAttr	x	(*exp*) Display properties for the Values axis (see DispAttr.*fontproperty* in the table)
Visible	x	(*exp*) Whether the graph control is visible
Width	x	(*exp*) The width of the graph
X	x	(*exp*) The x coordinate of the graph
Y	x	(*exp*) The y coordinate of the graph

Properties for GroupBox controls in DataWindow objects

An x in the M (Modify) column means you can change the property. When (*exp*) is included in the description, you can specify a DataWindow expression as the value for that property.

Property for a GroupBox	M	Description
Attributes		A list of the properties of the GroupBox control
Background.property	x	(*exp*) Background settings for the GroupBox control
Band		The band containing the GroupBox control
Border	x	(exp) Border style: 2 (box), 5 (3D lowered), 6 (3D raised)
Color	x	(*exp*) The text color
Font.property	x	(*exp*) Font settings for the text
Height	x	(*exp*) The height of the GroupBox control
HideSnaked	x	Whether the GroupBox control appears once per page when printing newspaper columns
Moveable	x	Whether the user can move the GroupBox control
Name		The name of the GroupBox control
Pointer	x	(*exp*) The pointer image when it is over the GroupBox control
Resizeable	x	Whether the user can resize the GroupBox control

Property for a GroupBox	M	Description
SlideLeft	x	(*exp*) Whether the GroupBox control moves left to fill in empty space
SlideUp	x	(*exp*) How the GroupBox control moves up to fill in empty space
Tag	x	(*exp*) The tag text for the GroupBox control
Text	x	(*exp*) The displayed text
Type		The control's type, which is GroupBox
Visible	x	(*exp*) Whether the GroupBox control is visible
Width	x	(*exp*) The width of the GroupBox control
X	x	(*exp*) The x coordinate of the GroupBox control
Y	x	(*exp*) The y coordinate of the GroupBox control

Properties for the Group keyword

You use these properties when generating DataWindow source code with the SyntaxFromSQL method.

Property	Description
NewPage (Group keywords)	Whether a change in a group column's value causes a page break
ResetPageCount	Whether a new value in a group column restarts page numbering

Properties for Line controls in DataWindow objects

An x in the M (Modify) column means you can change the property. When (*exp*) is included in the description, you can specify a DataWindow expression as the value for that property.

Property for a Line	M	Description
Attributes		A list of the properties of the line
Background.property	x	(*exp*) Background settings for the line
Band		The band containing the line
HideSnaked	x	Whether the control appears once per page when printing newspaper columns
Moveable	x	Whether the user can move the line
Name		The name of the line control
Pen.property	x	(*exp*) Appearance settings of the line
Pointer	x	(*exp*) The pointer image when it is over the line

Property for a Line	M	Description
Resizeable	x	Whether the user can resize the line
SlideLeft	x	(*exp*) Whether the line moves left to fill empty space
SlideUp	x	(*exp*) How the line moves up to fill empty space
Tag	x	(*exp*) The tag text for the line
Type		The control's type, which is Line
Visible	x	(*exp*) Whether the Line control is visible
X1, X2	x	(*exp*) The x coordinate of each end of the line
Y1, Y2	x	(*exp*) The y coordinate of each end of the line

Properties for OLE Object controls in DataWindow objects

An x in the M (Modify) column means you can change the property. When (*exp*) is included in the description, you can specify a DataWindow expression as the value for that property.

PocketBuilder
OLE Object controls are not supported in PocketBuilder.

Properties for Oval, Rectangle, and RoundRectangle controls in DataWindow objects

An x in the M (Modify) column means you can change the property. When (*exp*) is included in the description, you can specify a DataWindow expression as the value for that property.

Property	M	Description
Attributes		A list of the properties of the control
Background.property	x	(*exp*) Background settings for the control
Band		The band containing the control
Brush.property	x	(*exp*) Settings for fill pattern and color
Height	x	(*exp*) The height of the control
HideSnaked	x	Whether the control appears once per page when printing newspaper columns
Moveable	x	Whether the user can move the control
Name		The name of the control
Pen.property	x	(*exp*) Appearance settings of the control
Pointer	x	(*exp*) The pointer image when it is over the control

Property	M	Description
Resizeable	x	Whether the user can resize the control
SlideLeft	x	(*exp*) Whether the control moves left to fill empty space
SlideUp	x	(*exp*) How the control moves up to fill empty space
Tag	x	(*exp*) The tag text for the control
Type		The control's type, which is ellipse, rectangle, or roundrectangle
Visible	x	(*exp*) Whether the control is visible
X	x	(*exp*) The x coordinate of the control
Y	x	(*exp*) The y coordinate of the control

Additional properties for RoundRectangle controls in DataWindow objects

An x in the M (Modify) column means you can change the property. When (*exp*) is included in the description, you can specify a DataWindow expression as the value for that property.

Properties for Oval, Rectangle, and RoundRectangle controls in DataWindow objects also apply to RoundRectangle controls.

Property	M	Description
EllipseHeight	x	(*exp*) The radius of the vertical part of the rounded corner
EllipseWidth	x	(*exp*) The radius of the horizontal part of the rounded corner

Properties for Picture controls in DataWindow objects

An x in the M (Modify) column means you can change the property. When (*exp*) is included in the description, you can specify a DataWindow expression as the value for that property.

Property for a Picture	M	Description
Attributes		A list of the properties of the picture
Band		The band containing the picture
Border	x	(*exp*) The type of border around the picture
Filename	x	(*exp*) The file containing the picture
Height	x	(*exp*) The height of the picture
HideSnaked	x	Whether the control appears once per page when printing newspaper columns
HTML.property	x	(*exp*) Settings for creating a hyperlink for the picture

Property for a Picture	M	Description
Invert	x	(*exp*) Whether the colors are displayed inverted
Moveable	x	Whether the user can move the picture
Name		The name of the picture control
Pointer	x	(*exp*) The pointer image when it is over the picture
Resizeable	x	Whether the user can resize the picture
SlideLeft	x	(*exp*) Whether the picture moves left to fill in empty space
SlideUp	x	(*exp*) How the picture moves up to fill in empty space
Tag	x	(*exp*) The tag text for the picture
Type		The control's type, which is picture
Visible	x	(*exp*) Whether the picture control is visible
Width	x	(*exp*) The width of the picture
X	x	(*exp*) The x coordinate of the picture
Y	x	(*exp*) The y coordinate of the picture

Properties for Report controls in DataWindow objects

An x in the M (Modify) column means you can change the property. When (*exp*) is included in the description, you can specify a DataWindow expression as the value for that property.

Property for a Report	M	Description
Attributes		A list of the properties of the report
Band		The band containing the report
Border	x	(*exp*) The type of border around the report
Criteria	x	The search condition of the WHERE clause that relates the report to the main DataWindow
DataObject	x	The name of the DataWindow that is the nested report
Height	x	(*exp*) The height of the report
Height.AutoSize	x	Whether the height of the control will be adjusted to display all the data
HideSnaked	x	Whether the control appears once per page when printing newspaper columns
Moveable	x	Whether the user can move the report
Name		The name of the Report control
Nest_Arguments	x	Retrieval arguments for the report
NewPage (Report controls)	x	Whether to start the report on a new page (composite only)
Pointer	x	(*exp*) The pointer image when it is over the report

Property for a Report	M	Description
Resizeable	x	Whether the user can resize the report
SlideLeft	x	(*exp*) Whether the report moves left to fill in empty space
SlideUp	x	(*exp*) How the report moves up to fill in empty space
Tag	x	(*exp*) The tag text for the report
Trail_Footer	x	Where to print the footer (composite only)
Type		The control's type, which is report
Visible	x	(*exp*) Whether the Report control is visible
Width	x	(*exp*) The width of the report
X	x	(*exp*) The x coordinate of the report
Y	x	(*exp*) The y coordinate of the report

Properties for the Style keyword

You use these properties when generating DataWindow source code with the SyntaxFromSQL method.

Property	Description
Detail_Bottom_Margin	Bottom margin of the detail area
Detail_Top_Margin	Top margin of the detail area
Header_Bottom_Margin	Bottom margin of the header area
Header_Top_Margin	Top margin of the header area
Horizontal_Spread	Horizontal space between columns in the detail area
Left_Margin	The left margin of the DataWindow
Report	Whether the DataWindow is a read-only report
Type	The presentation style
Vertical_Size	The height of the columns in the detail area
Vertical_Spread	The vertical space between columns in the detail area

Properties for TableBlob controls in DataWindow objects

An x in the M (Modify) column means you can change the property. When (*exp*) is included in the description, you can specify a DataWindow expression as the value for that property.

Property for a TableBlob	M	Description
Attributes		A list of the properties of the TableBlob
Band		The band containing the TableBlob

Property for a TableBlob	M	Description
Border	x	(*exp*) The type of border around the TableBlob
ClientName	x	The name of the OLE client in the server window
Height	x	(*exp*) The height of the TableBlob
HideSnaked	x	Whether the control appears once per page when printing newspaper columns
ID		The number of the TableBlob
KeyClause	x	(*exp*) The key clause used when retrieving the blob
Moveable	x	Whether the user can move the TableBlob
Name		The name of the TableBlob
Pointer	x	(*exp*) The pointer image when it is over the TableBlob
Resizeable	x	Whether the user can resize the TableBlob
SlideLeft	x	(*exp*) Whether the TableBlob moves left to fill empty space
SlideUp	x	(*exp*) How the TableBlob moves up to fill empty space
Tag	x	(*exp*) The tag text for the control
Type		The control's type, which is TableBlob
Visible	x	(*exp*) Whether the TableBlob is visible
Width	x	(*exp*) The width of the TableBlob
X	x	(*exp*) The x coordinate of the TableBlob
Y	x	(*exp*) The y coordinate of the TableBlob

Properties for Text controls in DataWindow objects

An x in the M (Modify) column means you can change the property. An x in the S column means you can use the property with the SyntaxFromSQL method. When (*exp*) is included in the description, you can specify a DataWindow expression as the value for that property.

Property for text	M	S	Description
Alignment	x	x	The alignment of the text
Attributes			A list of the properties of the text control
Background.property	x	x	(*exp*) Background settings for the text control
Band			The band containing the text control
Border	x	x	(*exp*) The type of border around the text control
Color	x	x	(*exp*) The text color
Font.property	x	x	(*exp*) Font settings for the text
Height	x		(*exp*) The height of the text control
Height.AutoSize	x		Whether the control's height is adjusted to fit the data

Property for text	M	S	Description
HideSnaked	x		Whether the control appears once per page when printing newspaper columns
HTML.property	x		(*exp*) Settings for creating a hyperlink for the text
Moveable	x		Whether the user can move the text control
Name			The name of the text control
Pointer	x		(*exp*) The pointer image when it is over the text control
Resizeable	x		Whether the user can resize the text control
SlideLeft	x		(*exp*) Whether the text control moves left to fill space
SlideUp	x		(*exp*) How the text control moves up to fill empty space
Tag	x		(*exp*) The tag text for the text control
Text	x		(*exp*) The displayed text
Type			The control's type, which is Text
Visible	x		(*exp*) Whether the control is visible
Width	x		(*exp*) The width of the text control
X	x		(*exp*) The x coordinate of the text control
Y	x		(*exp*) The y coordinate of the text control

Title keyword

You use this property when generating DataWindow source code with the SyntaxFromSQL method.

Property	Description
Title("string")	The title for the DataWindow

Alphabetical list of DataWindow object properties

The properties for DataWindow objects and controls within a DataWindow object follow in alphabetical order.

To see the properties organized by type of control or syntax keyword, see "Controls in a DataWindow and their properties" on page 128.

Accelerator

Description

The accelerator key that a user can press to select a column in the DataWindow object.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✓

Applies to

Column controls

Syntax

PocketBuilder dot notation:

dw_control.Object.*columnname*.Accelerator

Describe and Modify argument:

"*columnname*.Accelerator { = '*acceleratorkey* ' }"

Parameter	Description
columnname	The name of the column for which you want to get or set the accelerator key.
acceleratorkey	(*exp*) A string expression whose value is the letter that will be the accelerator key for *columnname. Acceleratorkey* can be a quoted DataWindow expression.

Usage

An accelerator key for a column allows users to select a column (change focus) with a keystroke rather than with the mouse. The user changes focus by pressing the accelerator key in combination with the ALT key.

In the painter Select the control and set the value in the Properties view, Edit tab.

Displaying the accelerator The column does not display the key. To let users know what key to use, you can include an underlined letter in a text control that labels the column.When you enter the text control's label, precede the character you want underlined with an ampersand (&).

Accelerator keys and edit styles To use an accelerator key with the CheckBox or RadioButton edit style, select the Edit edit style and specify the accelerator there.

Examples

```
dw_1.Object.emp_name.Accelerator = 'A'
ls_data = dw_1.Describe("emp_name.Accelerator")
dw_1.Modify("emp_name.Accelerator='A'")
```

Action

Description

The action a user can assign to a button control.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Button controls

Syntax

PocketBuilder dot notation:

dw_control.Object.*buttonname*.Action

Describe and Modify argument:

"*buttonname*.Action { = ' *value* ' }"

Parameter	Description
buttonname	The name of the button for which you want to assign an action.
value	The action value assigned to the button. Values are listed in the following table.

Value	Action	Description	Value returned to ButtonClicked event
11	AppendRow	Inserts row at the end.	Row number of newly inserted row.
3	Cancel	Cancels a retrieval that has been started with the option to yield.	0
10	DeleteRow	If button is in detail band, deletes row associated with button; otherwise, deletes the current row.	1 if successful. -1 if an error occurs.
9	Filter	Displays Filter dialog box and filters as specified.	Number of rows filtered. Number < 0 if an error occurs.
12	InsertRow	If button is in detail band, inserts row using row number associated with the button; otherwise, inserts row using the current row.	Row number of newly inserted row.
6	PageFirst	Scrolls to the first page.	1 if successful. -1 if an error occurs.

Value	Action	Description	Value returned to ButtonClicked event
7	PageLast	Scrolls to the last page.	The row displayed at the top of the DataWindow control when the scrolling is complete or attempts to go past the first row. -1 if an error occurs.
4	PageNext	Scrolls to the next page.	The row displayed at the top of the DataWindow control when the scrolling is complete or attempts to go past the first row. -1 if an error occurs.
5	PagePrior	Scrolls to the prior page.	The row displayed at the top of the DataWindow control when the scrolling is complete or attempts to go past the first row. -1 if an error occurs.
16	Preview	Toggles between preview and print preview.	0
17	PreviewWith Rulers	Toggles between rulers on and off.	0
15	Print	Prints one copy of the DataWindow object.	0
20	QueryClear	Removes the WHERE clause from a query (if one was defined).	0
18	QueryMode	Toggles between query mode on and off.	0
19	QuerySort	Specifies sorting criteria (forces query mode on).	0
2	Retrieve	Retrieves rows from the database. The option to yield is not automatically turned on.	Number of rows retrieved.

Value	Action	Description	Value returned to ButtonClicked event
1	Retrieve (Yield)	Retrieves rows from the database. Before retrieval actually occurs, option to yield is turned on. This allows the Cancel action to take effect during a long retrieve.	Number of rows retrieved.
14	SaveRowsAs	Displays Save As dialog box and saves rows in the format specified.	Number of rows filtered.
8	Sort	Displays Sort dialog box and sorts as specified.	1 if successful. -1 if an error occurs.
13	Update	Saves changes to the database. If the update is successful, a COMMIT is issued. If the update fails, a ROLLBACK is issued	1 if successful. -1 if an error occurs.
0	UserDefined	(Default) Allows for programming of the ButtonClicked and ButtonClicking events with no intervening action occurring.	Return code from the user's coded event script.

Usage

In the painter Select the control and set the value in the Properties view, General tab.

Examples

```
dw_1.Object.b_retrieve.Action = "2"

setting = dw_1.Describe("b_retrieve.Action")

dw_1.Modify("b_retrieve.Action = '2'")
```

Activation

Description

The way the server for the OLE object in the OLE Object control is activated. Choices include letting the user activate the object by double-clicking or putting activation under program control.

PocketBuilder	✗
PowerBuilder	✓

Applies to

OLE Object controls

Syntax

PocketBuilder dot notation:

dw_control.Object.*olecontrolname*.Activation

Describe and Modify argument:

"*olecontrolname*.Activation { = ' *activationtype* ' }"

Alignment

Description

The alignment of the control's text within its borders.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Column, Computed Field, and Text controls

Syntax

PocketBuilder dot notation:

dw_control.Object.*controlname*.Alignment

Describe and Modify argument:

"*controlname*.Alignment { = ' *alignmentvalue* ' }"

SyntaxFromSQL:

Text (... Alignment = *value* ...)

Parameter	Description
controlname	The name of the control for which you want to get or set the alignment.
alignmentvalue	(*exp*) A number specifying the type of alignment for the text of *controlname*. *Alignmentvalue* can be a quoted DataWindow expression. Values are: 0 — (Default) Left 1 — Right 2 — Center 3 — Justified When generating DataWindow syntax with SyntaxFromSQL, the setting for Alignment applies to all text controls used as column labels.

Usage

When you select justified, the last line of text is not stretched to fill the line. Controls with only one line of text look left aligned.

In the painter Select the control and set the value using the:

- Properties view, General tab
- StyleBar

Examples

```
dw_1.Object.emp_name_t.Alignment = 2
ls_data = dw_1.Describe("emp_name.Alignment")
dw_1.Modify("emp_name_t.Alignment='2'")
```

Arguments

Description

The retrieval arguments required by the data source. You specify retrieval arguments in the DataWindow's SELECT statement and you provide values for the retrieval arguments when you call the Retrieve function.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Database table for the DataWindow object

Not settable in PowerScript. Used in DataWindow syntax.

Syntax

Table(Arguments = ((*name1*, *type*), (*name2*, *type*) ...) ...)

Parameter	Description
name	The name of the retrieval argument.
type	The type of the argument: • Date or a Date list • DateTime or a DateTime list • Number or a Number list • String or a String list • Time or a Time list

Usage

In the painter Set the value in the SQL painter.

Open the SQL painter by selecting Design>Data Source from the menu bar. Then select Design>Retrieval Arguments.

Attributes

Description

A tab-separated list of all the properties that apply to a control.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

DataWindow, Button, Column, Computed Field, Graph, GroupBox, Line, OLE, Oval, Picture, Rectangle, Report, RoundRectangle, TableBlob, and Text controls

Syntax

PocketBuilder dot notation:

dw_control.Object.*controlname*.Attributes

Describe argument:

"*controlname*.Attributes"

Examples

```
ls_data = dw_1.Object.emp_name_t.Attributes
ls_data = dw_1.Describe("DataWindow.Attributes")
ls_data = dw_1.Describe("emp_name_t.Attributes")
```

Axis

Description

The list of items or the expression associated with an axis of a graph. Each item is separated by a comma. You can ask for the list of categories on the Category axis, the series on the Series axis, or the values on the Values axis.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Graph controls

Syntax

PocketBuilder dot notation:

dw_control.Object.*graphname*.*axis*

Describe and Modify argument:

"*graphname*.*axis* { = ' *list* ' }"

Parameter	Description
graphname	The name of the graph within the DataWindow object for which you want to get or set the list of items for *axis*.
axis	An axis name. Values are: • Category • Series • Values
list	A string listing the categories, series, or values for the graph. The content of the list depends on the axis you specify. The items in the list are separated by commas. List is quoted.

Usage

In the painter Select the graph control and set the value by selecting a column or expression for each axis in the Properties view, Data tab.

Examples

```
ls_data = dw_1.Object.gr_1.Values
dw_1.Object.gr_1.Series = "Actual, Budget"
ls_data = dw_1.Describe("gr_1.Category")
ls_data = dw_1.Describe("gr_1.Series")
ls_data = dw_1.Describe("gr_1.Values")
dw_1.Modify("gr_1.Series='Actual, Budget'")
```

Axis.property

Description

Settings that control the appearance of an axis on a graph.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Graph controls

Syntax

PocketBuilder dot notation:

dw_control.Object.*graphname*.*axis*.*property*

Describe and Modify argument:

"*graphname*.*axis*.*property* { = *value* }"

Parameter	Description
graphname	The name of the graph within the DataWindow object for which you want to get or set a property value for an axis.
axis	An axis name. Values are: • Category • Series • Values
property	A property for the axis. Properties and their settings are listed in the table that follows.
value	The value to be assigned to the property. For axis properties, *value* can be a quoted DataWindow expression.

Property for Axis	Value
AutoScale	(*exp*) A boolean number specifying whether PocketBuilder scales the axis automatically. Values are: 0 — No, do not automatically scale the axis. 1 — Yes, automatically scale the axis. Painter: Axis tab, Scale group. Enabled when the axis displays nonstring data.
DispAttr.*fontproperty*	(*exp*) Properties that control the appearance of the text that labels the axis divisions. For a list of font properties, see the main entry for DispAttr.*fontproperty*. Painter: Text tab. Choose Category Axis Text, Series Axis Text, or Values Axis Text and set font properties.
DisplayEveryNLabels	(*exp*) An integer specifying which major axis divisions to label. For example, 2 means label every other tick mark. Values 0 and 1 both mean label every tick mark. If the labels are too long, they are clipped. Painter: Axis tab, Major Divisions group (not available for all graph types).

Property for Axis	Value
DropLines	(*exp*) An integer indicating the type of drop line for the axis. Values are: 0 — None 1 — Solid 2 — Dash 3 — Dot 4 — DashDot 5 — DashDotDot Painter: Axis tab, Major Divisions group (not available for all graph types).
Frame	(*exp*) An integer indicating the type of line used for the frame. Values are 0–5. See DropLines in this table for their meaning. Painter: Axis tab, Line Style group (available for 3D graph types).
Label	(*exp*) A string whose value is the axis label. Painter: Axis tab.
LabelDispAttr. *fontproperty*	(*exp*) Properties that control the appearance of the axis label. For a list of font properties, see the main entry for DispAttr.*fontproperty*. Painter: Text tab. Choose Category Axis Label, Series Axis Label, or Values Axis Label and set font properties.
MajorDivisions	(*exp*) An integer specifying the number of major divisions on the axis. Painter: Axis tab, Major Divisions group.
MajorGridLine	(*exp*) An integer specifying the type of line for the major grid. Values are 0–5. See DropLines in this table for their meaning. Painter: Axis tab, Major Divisions group.
MajorTic	(*exp*) An integer specifying the type of the major tick marks. Values are: 1 — None 2 — Inside 3 — Outside 4 — Straddle Painter: Axis tab, Major Divisions group.
MaximumValue	(*exp*) A double specifying the maximum value for the axis. Painter: Axis tab, Scale group.
MinimumValue	(*exp*) A double specifying the minimum value for the axis. Painter: Axis tab, Scale group.

Property for Axis	Value
MinorDivisions	(*exp*) An integer specifying the number of minor divisions on the axis. Painter: Axis tab, Minor Divisions group.
MinorGridLine	(*exp*) An integer specifying the type of line for the minor grid. Values are 0–5. See DropLines in this table for their meaning. Painter: Axis tab, Minor Divisions group.
MinorTic	(*exp*) An integer specifying the type of the minor tick marks. Values are: 1 — None 2 — Inside 3 — Outside 4 — Straddle Painter: Axis tab, Minor Divisions group.
OriginLine	(*exp*) An integer specifying the type of origin line for the axis. Values are 0–5. See DropLines in this table for their meaning. Painter: Axis tab, Line Style group. Enabled for numeric data axes.
PrimaryLine	(*exp*) An integer specifying the type of primary line for the axis. Values are 0–5. See DropLines in this table for their meaning. Painter: Axis tab, Line Style group.
RoundTo	(*exp*) A double specifying the value to which you want to round the axis values. Specify both a value and a unit (described next). Painter: Axis tab, Scale group.
RoundToUnit	(*exp*) An integer specifying the units for the rounding value. The units must be appropriate for the axis datatype. Values are: 0 — Default, for an axis of any datatype 1 — Years, for an axis of type date or DateTime 2 — Months, for an axis of type date or DateTime 3 — Days, for an axis of type date or DateTime 4 — Hours, for an axis of type time or DateTime 5 — Minutes, for an axis of type time or DateTime 6 — Seconds, for an axis of type time or DateTime 7 — Microseconds, for an axis of type time or DateTime Painter: Axis tab, Scale group.

Property for Axis	Value
ScaleType	(*exp*) An integer specifying the type of scale used for the axis. Values are: 1 — Scale_Linear 2 — Scale_Log10 3 — Scale_Loge Painter: Axis tab, Scale group.
ScaleValue	(*exp*) An integer specifying the scale of values on the axis. Values are: 1 — Scale_Actual 2 — Scale_Cumulative 3 — Scale_Percentage 4 — Scale_CumPercent Painter: Axis tab, Line Style group.
SecondaryLine	(*exp*) An integer specifying the type of secondary line for the axis. The line is parallel to and opposite the primary line and is usually not displayed in 2D graphs. Values are 0–5. See DropLines in this table for their meaning. Painter: Axis tab, Line Style group.
ShadeBackEdge	(*exp*) A boolean number specifying whether the back edge of the axis is shaded. Values are: 0 — No, the back edge is not shaded 1 — Yes, the back edge is shaded Painter: Axis tab. Enabled for 3D graphs only.
Sort	(*exp*) An integer specifying the way the axis values should be sorted. (Does not apply to the Values axis.) Values are: 0 — Unsorted 1 — Ascending 2 — Descending Painter: Axis tab, Line Style group.

Usage

In the painter Select the graph control and set the value in the Properties view, various tabs. To set most axis properties, select the Axis tab and an axis in the Axis drop-down list.

Examples

```
string ls_data
ls_data = dw_1.Object.gr_1.Category.AutoScale

dw_1.Object.Category.LabelDispAttr.Alignment = 2

ls_data = dw_1.Describe("gr_1.Category.AutoScale")

dw_1.Modify("gr_1.Series.AutoScale=0")
```

```
dw_1.Modify("gr_1.Values.Label='Cities'")
dw_1.Modify("gr_1.Category.LabelDispAttr.Alignment=2")
```

BackColor

Description

The background color of a graph in a DataWindow.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Graph controls

Syntax

PocketBuilder dot notation:

dw_control.Object.*graphname*.BackColor

Describe and Modify argument:

"*graphname*.BackColor { = *long* }"

Parameter	Description
graphname	The graph whose background color you want to get or set.
long	(*exp*) A long expression specifying the color (red, green, and blue values) to be used as the graph's background color. *Long* can be a quoted DataWindow expression.

Usage

In the painter Select the graph control and set the value in the Properties view, General tab.

Examples

```
dw_1.Object.graph_1.BackColor = 250
setting = dw_1.Describe("graph_1.BackColor")
dw_1.Modify("graph_1.BackColor=250")
```

Background.*property*

Description

Settings for the color and transparency of a control.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Button, Column, Computed Field, GroupBox, Line, Oval, Rectangle, RoundRectangle, and Text controls

Syntax

PocketBuilder dot notation:

dw_control.Object.*controlname*.Background.*property*

Describe and Modify argument:

"*controlname*.Background.*property* { = ' *value* ' }"

SyntaxFromSQL:

Column (Background.*property* = *value*)

Text (Background.*property* = *value*)

Parameter	Description
controlname	The control whose Background properties you want to get or set. When generating DataWindow syntax with SyntaxFromSQL, the Background settings apply to all columns or all text controls.
property	A property that applies to the background of a control, as listed in the Property table below.
value	Values for the properties are shown below. *Value* can be a quoted DataWindow expression.

Property for Background	Value
Color	(*exp*) A long expression specifying the color (the red, green, and blue values) to be used as the control's background color.
Mode	(*exp*) A number expression specifying the mode of the background of *controlname*. Values are: 0 — Make the control's background opaque. 1 — Make the control's background transparent.

Usage

In the painter Select the control and set the value in the Properties view Font tab for controls that have text and in the General tab for drawing controls (choose Transparent or a color)

When you choose a Brush Hatch fill pattern other than Solid for an Oval, Rectangle, or RoundRectangle control, the Background Color and the Brush Color are used for the pattern colors.

Background color of a line The background color of a line is the color that displays between the segments of the line when the pen style is not solid.

Transparent background If Background.Mode is transparent (1), Background.Color is ignored.

DropDownDataWindows and GetChild When you set Background.Color and Background.Mode for a column with a DropDownDataWindow, references to the DropDownDataWindow become invalid. Call GetChild again after changing these properties to obtain a valid reference.

Examples

```
dw_1.Object.oval_1.Background.Color = RGB(255, 0, 128)

ls_data = dw_1.Describe("oval_1.Background.Color")

dw_1.Modify("emp_name.Background.Color='11665407'")

ls_data = dw_1.Describe("emp_name.Background.Mode")

dw_1.Modify("emp_name.Background.Mode='1'")

dw_1.Modify("rndrect_1.Background.Mode='0'")

SQLCA.SyntaxFromSQL(sql_syntax, &
    "Style(...) Column(Background.Mode=1 ...) ...", &
    ls_Errors)

SQLCA.SyntaxFromSQL(sql_syntax, &
    "Style(...) Column(Background.Color=11665407 ...)",
&
    ls_Errors)
```

Band

Description

The band or layer in the DataWindow object that contains the control. The returned text is one of the following, where # is the level number of a group: detail, footer, header, header.#, summary, trailer.#, foreground, background.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Changing a control's band
Use the SetPosition function to change a control's band during execution.

Applies to

Button, Column, Computed Field, Graph, GroupBox, Line, OLE, Oval, Picture, Rectangle, Report, RoundRectangle, TableBlob, and Text controls

Syntax

PocketBuilder dot notation:

dw_control.Object.*controlname*.Band

Describe and Modify argument:

"*controlname*.Band"

Parameter	Description
controlname	The name of the control within the DataWindow for which you want the band it occupies

Usage

In the painter Select the control and set the value in the Properties view, Position tab, Layer option. When the control's layer is Band, you can drag the control into another band.

Examples

```
ls_data = dw_1.Object.emp_title.Band
ls_data = dw_1.Describe("emp_title.Band")
```

Bandname.property

Description

Settings for the color, size, and pointer of a band in the DataWindow object.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

DataWindows

Syntax

PocketBuilder dot notation:

dw_control.Object.DataWindow.*bandname*.*property*

Describe and Modify argument:

"DataWindow.*bandname*{.#}.*property* { = *value* }"

Parameter	Description
bandname	The identifier of a band in the DataWindow object. Values are: • Detail • Footer • Summary • Header • Trailer **Setting the header.# and trailer.# bands** You cannot use dot notation to set the header.# and trailer.# bands.
#	The number of the group you want when *bandname* is Header or Trailer. The group must exist.
property	A property that applies to the band, as listed in the table below.
value	Values for the properties are shown in the following table.

Property for Bandname	Value
Color	(*exp*) A long specifying the color (the red, green, and blue values) to be used as the band's background color. *Value* can be a quoted DataWindow expression. Painter: General tab.
Height	An integer specifying the height of the detail area in the unit of measure specified for the DataWindow. Painter: General tab. For another way of setting the height of the detail band, see the SetDetailHeight function.

Property for Bandname	Value
Height.AutoSize	(Only when *bandname* is Detail) Allows the band to grow to display the entire content of a row. Selecting this property sets the minimum height for all rows to the size specified by the Height property for the band. Values are: No — Make all the row heights the same. Yes — Expand the row height to accommodate the row content and set a minimum size for all rows in the Detail band. This property can be especially useful for viewing the contents of a row in which the Height.AutoSize property is set on a text column in the row. The height of the detail band must not grow larger than a page, except for bands containing nested DataWindows with the Report.Height.AutoSize property set to Yes. Painter: General tab when the Detail band is selected.
Pointer	(*exp*) A string specifying a value of the Pointer enumerated datatype or the name of a cursor file (.CUR) to be used for the pointer. See the SetPointer function for a list of Pointer values. *Pointername* can be a quoted DataWindow expression. Painter: Pointer tab.

Usage

In the painter Select the band by clicking the gray divider for the band. Set the value in the Properties view.

Examples

```
string ls_data
ls_data = dw_1.Object.DataWindow.Detail.Height

dw_1.Object.DataWindow.Detail.Pointer = "hand.cur"

ls_data = dw_1.Describe("DataWindow.Detail.Height")

ls_data = &
    dw_1.Describe("DataWindow.Detail.Height.AutoSize")

dw_1.Modify("DataWindow.Detail.Pointer='hand.cur'")

dw_1.Modify("DataWindow.Detail.Pointer=' ~"Cross!~" ~t
&
if(emp_status=~"a~", ~"HourGlass!~", ~"Cross!~")'")

dw_1.Modify("DataWindow.Footer.Height=250")

ll_color = RGB(200, 200, 500)
dw_1.Modify("DataWindow.Header.2.Color=" &
    + String(ll_color))
```

```
dw_1.Modify("DataWindow.Trailer.2.Height=500")

dw_1.Modify( &
"DataWindow.Summary.Pointer='c:\pb\total.cur'")
```

Bandname.Text

Description

(RichText presentation style only) The rich text content of the specified band as an ASCII string.

PocketBuilder	✗
PowerBuilder	✓

When you use Describe or dot notation, nested quotes are converted to tilde-quote combinations. To get pure RTF data, use the CopyRTF function.

Applies to

DataWindows in the RichText presentation style

Syntax

PowerBuilder dot notation:

dw_control.Object.DataWindow.*bandname*.Text

Describe and Modify argument:

"DataWindow.*bandname*.Text { = *rtfstring* }"

Bands

Description

A list of the bands in the DataWindow object. The list can include one or more of the following band identifiers, where # is the level number of a group: Detail, Footer, Header, Header.#, Summary, Trailer.#. The items in the list are separated by tabs.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

DataWindows

Syntax

PocketBuilder dot notation:

dw_control.Object.DataWindow.Bands

Describe argument:

"DataWindow.Bands"

Examples

```
ls_data = dw_1.Object.DataWindow.Bands
ls_data = dw_1.Describe("DataWindow.Bands")
```

BinaryIndex

Description

An internal index that PowerBuilder uses to manage the OLE Object control in the library. But there is no reason to get this value; the value has no external significance.

PocketBuilder	✗
PowerBuilder	✓

Applies to

OLE Object controls

Syntax

"*olecontrolname*.BinaryIndex"

BitmapName

Description

Whether PocketBuilder interprets the column's value as the name of a picture file and displays the picture instead of the text. BitmapName's value is either Yes or No.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Column controls

Syntax

PocketBuilder dot notation:

dw_control.Object.*columnname*.BitmapName

Describe argument:

"*columnname*.BitmapName"

Usage

In the painter Select the control and set the value in the Properties view, General tab, Display As Pic option.

Examples

```
ls_data = dw_1.Object.emp_name.BitmapName
ls_data = dw_1.Describe("emp_name.BitmapName")
```

Border

Description

The type of border for the control.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

PocketBuilder
The value 3 (Resize) is not supported in PocketBuilder applications.

Applies to

Column, Computed Field, Graph, GroupBox, OLE, Picture, Report, TableBlob, and Text controls

Syntax

PocketBuilder dot notation:

dw_control.Object.*controlname*.Border

Describe and Modify argument:

"*controlname*.Border { = ' *value* ' }"

SyntaxFromSQL:

Column (... Border = *value ...*)

Text (... Border = *value ...*)

Parameter	Description
controlname	The name of the control whose border you want to get or set. When generating DataWindow syntax with SyntaxFromSQL, the Border setting applies to all columns or all text controls.
value	(*exp*) A number specifying the type of border. Values are: 0 — None 1 — Shadow 2 — Rectangle 3 — Resize 4 — Line 5 — 3D Lowered 6 — 3D Raised Integer can be a DataWindow quoted painter expression. When you change between Resize and another border, change the Resizeable property too so that the control's appearance and behavior match. For columns, you can access the Border property with the GetBorderStyle and SetBorderStyle functions.

Usage

In the painter Select the control and set the value in the Properties view, General tab.

Changing the Border setting between Resize and another border affects the Resizeable option on the Position tab. To make another border resizable, choose the border. Close and then redisplay the property sheet and check Resizeable on the Position tab.

For examples of other ways to set properties, using Border as an example, see "What you can do with DataWindow object properties" on page 345.

Examples

```
string ls_data
ls_data = dw_1.Object.emp_name_t.Border

dw_1.Object.emp_name_t.Border='6'

ls_data = dw_1.Describe("emp_name_t.Border")

dw_1.Modify("emp_name_t.Border='6'")

SQLCA.SyntaxFromSQL(sql_syntax, &
       "Style(...) Column(Border=5 ...) ...",
ls_Errors)
```

Brush.*property*

Description

Settings for the fill pattern and color of a graphic control.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

PocketBuilder
The Brush.Hatch property is not supported in PocketBuilder applications.

Applies to

Oval, Rectangle, and RoundRectangle controls

Syntax

PocketBuilder dot notation:

dw_control.Object.*controlname*.Brush.*property*

Describe and Modify argument:

"*controlname*.Brush.*property* { = ' *value* ' }"

Parameter	Description
controlname	The name of the Line, Oval, Rectangle, RoundRectangle, or Text control whose Brush property you want to get or set.
property	A property that applies to the Brush characteristics of a control, as listed in the table below.
value	Values for the properties are shown below. Value can be a quoted DataWindow expression.

Property for Brush	Value
Color	(*exp*) A long expression specifying the color (the red, green, and blue values) to be used to fill the control.
Hatch	(*exp*) A number expression specifying the fill pattern of *controlname*. Values are: 0 — Horizontal 1 — Bdiagonal (lines from lower left to upper right) 2 — Vertical 3 — Cross 4 — Fdiagonal (lines from upper left to lower right) 5 — DiagCross 6 — Solid 7 — Transparent

Usage

In the painter Select the control and set the value in the Properties view, General tab.

When you choose a Brush Hatch fill pattern other than Solid, the Background Color and the Brush Color are used for the pattern colors.

Examples

```
string ls_data
ls_data = dw_1.Object.oval_1.Brush.Hatch

dw_1.Object.oval_1.Brush.Hatch = 5

ls_data = dw_1.Describe("oval_1.Brush.Hatch")

dw_1.Modify("oval_1.Brush.Hatch='5'")

dw_1.Modify("oval_1.Brush.Color='16731766'")
```

Category

See Axis, Axis.property, and DispAttr.fontproperty.

CheckBox.*property*

Description

Settings for a column whose edit style is CheckBox.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Column controls

Syntax

PocketBuilder dot notation:

dw_control.Object.*columnname*.CheckBox.*property*

Describe and Modify argument:

"*columnname*.CheckBox.*property* { = *value* }"

Parameter	Description
columnname	The column whose edit style is CheckBox for which you want to get or set property values.
property	A property for the CheckBox edit style, as listed in the table below.
value	Values for the properties are shown in the table below. For CheckBox properties, *value* cannot be a DataWindow expression.

Property for CheckBox	Value
3D or ThreeD	Whether the CheckBox should be 3D. Values are: Yes — Make the CheckBox 3D No — Do not make the CheckBox 3D Painter: 3D Look option. **Setting the Checkbox 3D property** When using dot notation, use the term ThreeD instead of 3D.
LeftText	Whether the CheckBox label is to the left or right of the CheckBox. Values are: Yes — Display the label on the left. No — Display the label on the right. Painter: Left Text option.

Property for CheckBox	Value
Off	A string constant specifying the column value when the CheckBox is off (unchecked). The resulting value must be the same datatype as the column. Painter: Data Value for Off option.
On	A string constant specifying the value that will be put in the column when the CheckBox is on (checked). The resulting value must be the same datatype as the column. Painter: Data Value for On option.
Other	A string constant specifying the value that will be put in the column when the CheckBox is in the third state (neither checked nor unchecked). The value must be the same datatype as the column. Painter: Other State option is available when 3 States is checked.
Scale	Whether you want to scale the 2D CheckBox. Takes effect only when the 3D property is No. Values are: Yes — Scale the CheckBox No — Do not scale the CheckBox Painter: Scale option.
Text	A string specifying the CheckBox's label text. Painter: Text option.

Usage

In the painter Select the control and set values in the Properties view, Edit tab, when Style Type option is CheckBox.

Examples

```
dw_1.Object.emp_gender.CheckBox.ThreeD = "no"

IF dw_1.Object.emp_status.CheckBox.LeftText = "yes"
THEN
dw_1.Object.emp_status2.CheckBox.LeftText = "yes"
END IF

dw_1.Modify("emp_gender.CheckBox.3D=no")

IF dw_1.Describe("emp_status.CheckBox.LeftText") &
       = "yes" THEN
dw_1.Modify("emp_status2.CheckBox.LeftText=yes")
END IF

dw_1.Modify("emp_status.CheckBox.Off='Terminated'")

dw_1.Modify("emp_status.CheckBox.On='Active'")

dw_1.Modify("emp_status.CheckBox.Other='Unknown'")
```

ClientName

Description

The name of the OLE client. The default is "Untitled." ClientName is used by some applications in the server window's title.

PocketBuilder	✗
PowerBuilder	✓

Applies to

OLE Object and TableBlob controls

Syntax

PowerBuilder dot notation:

dw_control.Object.*controlname*.ClientName

Describe and Modify argument:

"*controlname*.ClientName { = ' *clientname* ' }"

Color

Description

The text color of the column or the background color of the DataWindow.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

The color affected by the Color property depends on the control:

- For the DataWindow, Color specifies the background color
- For columns, computed fields, and text, Color specifies the text color
- For graphs, Color specifies the line color, used for axes, borders around data markers, tickmarks, and the outline of the box for 3D graphs

Applies to

DataWindow, Button, Column, Graph, and GroupBox controls

Syntax

PocketBuilder dot notation:

dw_control.Object.DataWindow.Color

dw_control.Object.*controlname*.Color

Describe and Modify argument:

"DataWindow.Color { = *long* }"

"*controlname*.Color { = *long* }"

SyntaxFromSQL:

DataWindow (Color = *long*)

Column (Color = *long*)

Parameter	Description
controlname	The column whose text color you want to set or the graph whose line color you want to set.
long	(*exp* for columns only) A long value specifying the color of the column text or the DataWindow background. When you are specifying the text color of a column, you can specify a DataWindow expression in quotes. You cannot specify an expression for the DataWindow background color. When generating DataWindow syntax with SyntaxFromSQL, the Color setting for Column applies to all columns.

Usage

In the painter For the DataWindow background, click the DataWindow to deselect all controls and set the value in the Properties view, General tab, Color option.

For a column's text color, select the column and set the value in the Properties view, Font tab, Text Color option.

For a graph's line color, select the graph and set the value in the Properties view, General tab, Text Color option.

Examples

```
string column_text_color
column_text_color = dw_1.Object.emp_name.Color

dw_1.Object.salary.Color = &
       "0~tIf(salary>90000,255,65280)"

dw_back_color = dw_1.Describe("DataWindow.Color")

column_text_color = dw_1.Describe("emp_name.Color")

dw_1.Modify( &
       "salary.Color='0~tIf(salary>90000,255,65280)'")
```

See also

Background, BackColor

ColType

Description

The datatype of the column or computed field.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Column and Computed Field controls

Syntax

PocketBuilder dot notation:

dw_control.Object.*controlname*.ColType

Describe argument:

"*controlname*.ColType"

Parameter	Description
controlname	The column for which you want the datatype. Possible datatypes are: • Char (*n*) — *n* is the number of characters • Date • DateTime • Decimal (*n*) — *n* is the number of decimal places • Int • Long • Number • Real • Time • Timestamp • ULong

Usage

In the painter The value of ColType is derived from the data or expression you specify for the control. The value is displayed in the Column Specifications view.

Date column types
If you define a DataWindow with a column of type Date and deploy it with a DBMS that uses the DateTime datatype, set the StaticBind DBParm parameter to 0 or No. This forces PocketBuilder to get a result set description before retrieving data and adjust the bind information if necessary.

For more information, see the StaticBind DBParm parameter in the online Help.

Examples

```
string ls_coltype
ls_coltype = dw_1.Object.emp_id.ColType

ls_coltype = dw_1.Describe("emp_id.ColType")
```

Column.Count

Description

The number of columns in the DataWindow object.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

DataWindows

Syntax

PocketBuilder dot notation:

dw_control.Object.DataWindow.Column.Count

Describe argument:

"DataWindow.Column.Count"

Usage

In the painter The value is determined by the number of columns you select in the Select painter, whether or not they are displayed.

Examples

```
string ls_colcount
ls_colcount = dw_1.Object.DataWindow.Column.Count

ls_colcount = dw_1.Describe("DataWindow.Column.Count")
```

ContentsAllowed

Description

The way the OLE Object control holds the OLE object. You can restrict the container to only embedded or only linked objects, or you can allow either type.

PocketBuilder	✗
PowerBuilder	✓

Applies to

OLE Object controls

Syntax

PowerBuilder dot notation:

dw_control.Object.*olecontrolname*.ContentsAllowed

Describe and Modify argument:

"*olecontrolname*.ContentsAllowed { = ' *contentstype* ' }"

Criteria

Description

The search condition of the WHERE clause for a related report. The Criteria property defines the connection between the related report and the DataWindow.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Report controls

Syntax

PocketBuilder dot notation:

dw_control.Object.*reportname*.Criteria

Describe and Modify argument:

"*reportname*.Criteria { = *string* }"

Parameter	Description
reportname	The name of the report control for which you want to get or set Criteria.
string	An expression that will be the search condition of the WHERE clause for the related report.

Examples

```
ls_colcount = dw_1.Object.rpt_1.Criteria
```

```
dw_1.Object.rpt_1.Criteria = "emp_id=:emp_id"
ls_colcount = dw_1.Describe("rpt_1.Criteria")
dw_1.Modify("rpt_1.Criteria='emp_id=:emp_id'")
```

See also Nest_Arguments DataWindow object property

Criteria.*property*

Description Settings for the Prompt for Criteria dialog box. When Prompt for Criteria is enabled, PocketBuilder prompts the user to specify criteria for retrieving data whenever the Retrieve function is called. Note that the Required property also affects query mode.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax PocketBuilder dot notation:

dw_control.Object.*columnname*.Criteria.*property*

Describe and Modify argument:

"*columnname*.Criteria.*property* { = *value* }"

Parameter	Description
columnname	The name of the column for which you want to get or set Prompt for Criteria properties.
property	A property for the Prompt for Criteria dialog. Properties and their settings are listed in the table below.
value	A Yes or No value to be assigned to the property. For Criteria properties, *value* cannot be a DataWindow expression.

Property for Criteria	Value
Dialog	Whether Prompt for Criteria is on for *columnname*. Values are: Yes — Include *columnname* in the Prompt for Criteria dialog box. No — (Default) Do not include *columnname* in the Prompt for Criteria dialog box. If the Dialog property is Yes for at least one column in the DataWindow, then PocketBuilder displays the Prompt for Criteria dialog box when the Retrieve function is called. Painter: Column Specifications view, Prompt checkbox.
Override_Edit	Whether the user must enter data in the Prompt for Criteria dialog box according to the edit style defined for the column in the DataWindow object or be allowed to enter any specifications in a standard edit control. Values are: Yes — Allow the user to override the column's edit style and enter data in a standard edit control. No — (Default) Constrain the user to the edit style for the column. Painter: Properties view, General tab, Override Edit option.
Required	Whether the user is restricted to the equality operator (=) when specifying criteria in query mode and in the Prompt for Criteria dialog box. Values are: Yes — Require the user to use the equality operator only. No — (Default) Allow the user to use any relational operator, including =, <>, <, >, >=, and <=. Painter: Properties view, General tab, Equality Required option.

Usage

In the painter Set the values using the menus and Properties view as described in the table above.

Examples

```
string setting
setting = dw_1.Object.empname.Criteria.Dialog

dw_1.Object.empname.Criteria.Dialog= "Yes"

setting = dw_1.Describe("empname.Criteria.Dialog")

dw_1.Modify("empname.Criteria.Dialog=Yes")

dw_1.Modify("empname.Criteria.Override_Edit=Yes")

dw_1.Modify("empname.Criteria.Required=No")
```

```
IF dw_1.Describe("empname.Edit.Style") &
       = "dddw" THEN
dw_1.Modify("empname.Criteria.Override_Edit=Yes")
END IF
```

Crosstab.*property*

Description

Settings for a DataWindow object whose presentation style is Crosstab.

PocketBuilder	✗
PowerBuilder	✓

Applies to

DataWindows

Syntax

PowerBuilder dot notation:

dw_control.Object.DataWindow.Crosstab.*property*

Describe and Modify argument:

"DataWindow.Crosstab.*property* { = *value* }"

Data

Description

A tab-separated list describing the data in the DataWindow object.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

DataWindows

Syntax

PocketBuilder dot notation:

dw_control.Object.DataWindow.Data

Describe argument:

"DataWindow.Data"

Examples

```
string setting
setting = dw_1.Object.DataWindow.Data

setting = dw_1.Describe("DataWindow.Data")
```

Data.HTML

Description

A string containing HTML and JavaScript that represents data and presentation of the DataWindow object.

PocketBuilder	✗
PowerBuilder	✓

The data is presented in a read-only HTML table or data-entry form depending on settings of other properties.

Applies to

DataWindows

Syntax

PowerBuilder dot notation:

dw_control.Object.DataWindow.Data.HTML

Describe argument:

"DataWindow.Data.HTML"

Data.HTMLTable

Description

The data in the DataWindow object described in HTML table format. This property is used in the process of dynamically creating Web pages from a database.

PocketBuilder	✗
PowerBuilder	✓

Applies to

DataWindows

Syntax

PowerBuilder dot notation:

dw_control.Object.DataWindow.Data.HtmlTable

Describe argument:

"DataWindow.Data.HtmlTable"

Data.XML

Description

A string containing the row data content of the DataWindow object in XML format.

PocketBuilder	✗
PowerBuilder	✓

Applies to

DataWindows

Syntax

PowerBuilder dot notation:

dw_control.Object.DataWindow.Data.XML

Describe argument:

"DataWindow.Data.XML"

Data.XMLDTD

Description

A string containing the full document type definition (DTD) of the XML output for a DataWindow object.

PocketBuilder	✗
PowerBuilder	✓

Applies to

DataWindows

Syntax

PowerBuilder dot notation:

dw_control.Object.DataWindow.Data.XMLDTD

Describe argument:

"DataWindow.Data.XMLDTD"

Data.XMLSchema

Description

A string containing the full schema of the XML output of a DataWindow object.

PocketBuilder	✗
PowerBuilder	✓

Applies to

DataWindows

Syntax

PowerBuilder dot notation:

dw_control.Object.DataWindow.Data.XMLSchema

Describe argument:

"DataWindow.Data.XMLSchema"

Data.XSLFO

Description

A string containing XSL Formatting Objects (XSL-FO) that represents the data and presentation of the DataWindow object.

PocketBuilder	✗
PowerBuilder	✓

Applies to

DataWindows

Syntax

PowerBuilder dot notation:

dw_control.Object.DataWindow.Data.XSLFO

Describe argument:

"DataWindow.Data.XSLFO"

DataObject

Description

The name of the DataWindow that is the nested report within the main DataWindow.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Report controls

Syntax

PocketBuilder dot notation:

dw_control.Object.*reportname*.DataObject

Describe and Modify argument:

"*reportname*.DataObject = ' *dwname* ' "

Parameter	Description
reportname	The name of the Report control in the main DataWindow for which you want to get or set the nested DataWindow.
dwname	A string naming a DataWindow object in the application's libraries that is the DataWindow for the report within the main DataWindow.

Usage

In the painter Select the control and set the value in the Properties view, General tab, Report option.

Examples

```
setting = dw_1.Object.rpt_1.DataObject
dw_1.Object.rpt_1.DataObject = "d_empdata"
setting = dw_1.Describe("rpt_1.DataObject")
dw_1.Modify("rpt_1.DataObject='d_empdata'")
```

dbName

Description

The name of the database column. PocketBuilder uses this value to construct the update syntax.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Column controls

Syntax

PocketBuilder dot notation:

dw_control.Object.*columnname*.dbName

Describe and Modify argument:

"*columnname*.dbName { = ' *dbcolumnname* ' }"

Parameter	Description
columnname	The name of the column for which you want the name of the corresponding database column.
dbcolumnname	The name of the database column associated with *columnname.*

Usage

DbName is the name of the database column in the format *tablename.columnname*. The value of dbName does not include the quotes that can be part of the SQL syntax.

In the painter The Syntax view displays the database column names (they can be shown with quotes).

Examples

```
dbcol = dw_1.Object.emp_id.dbName

dw_1.Object.emp_id.dbName = "emp_id"

dbcol = dw_1.Describe("emp_id.dbName")

dw_1.Modify("emp_id.dbName='emp_id'")
```

dddw.*property*

Description

Properties that control the appearance and behavior of a column with the DropDownDataWindow edit style.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Column controls

Syntax

PocketBuilder dot notation:

dw_control.Object.*columnname*.dddw.*property*

Describe and Modify argument:

"*columnname*.dddw.*property* { = *value* }"

Parameter	Description
columnname	The name of a column that has the DropDownDataWindow edit style.
property	A property for the DropDownDataWindow column. Properties and their settings are listed in the table below.
value	The value to be assigned to the property. For dddw properties, *value* cannot be a DataWindow expression.

Property for dddw	Value
AllowEdit	Whether the user can type a value as well as choose from the DropDownDataWindow's list. Values are: Yes — Typing is allowed. No — (Default) Typing is not allowed. Call GetChild *after* setting dddw.AllowEdit to get a valid reference to the column's DropDownDataWindow. Painter: Allow Editing option.
AutoHScroll	Whether the DropDownDataWindow automatically scrolls horizontally when the user enters or deletes data. Values are: Yes — (Default) Scroll horizontally automatically. No — Do not scroll automatically. Painter: Auto Horz Scroll option.
AutoRetrieve	Whether the DropDownDataWindow data is retrieved when the parent DataWindow data is retrieved. Values are: Yes — (Default) Data is automatically retrieved. No — Data must be retrieved separately. Painter: AutoRetrieve option.
Case	The case of the text in the DropDownDataWindow. Values are: Any — Character of any case allowed. Upper — Characters converted to uppercase. Lower — Characters converted to lowercase. Call GetChild *after* setting dddw.Case to get a valid reference to the column's DropDownDataWindow. Painter: Case option.
DataColumn	A string whose value is the name of the data column in the associated DropDownDataWindow. *Value* is quoted. Call GetChild *after* setting dddw.DataColumn to get a valid reference to the column's DropDownDataWindow. Painter: Data Column option, visible after selecting a DataWindow.

Property for dddw	Value
DisplayColumn	A string whose value is the name of the display column in the associated DropDownDataWindow. *Value* is quoted. Call GetChild *after* setting dddw.DisplayColumn to get a valid reference to the column's DropDownDataWindow. Painter: Display Column option, visible after selecting a DataWindow.
HScrollBar	Whether a horizontal scroll bar displays in the DropDownDataWindow. Values are: Yes — Display a horizontal scroll bar. No — Do not display a horizontal scroll bar. Painter: H ScrollBar option.
HSplitScroll	Whether the horizontal scroll bar is split. The user can adjust the split position. Values are: Yes — Split the horizontal scroll bar so the user can scroll the display and data columns separately. No — The horizontal scroll bar is not split. Painter: Split Horz Scroll Bar option.
Limit	An integer from 0 to 32767 specifying the maximum number of characters that can be entered in the DropDownDataWindow. Zero means unlimited. Painter: Limit option.
Lines	An integer from 0 to 32767 specifying the number of lines (values) to display in the DropDownDataWindow. Painter: Lines in DropDown option.
Name	A string whose value is the name of the DropDownDataWindow associated with the column. Call GetChild *after* setting dddw.Name to get a valid reference to the column's DropDownDataWindow. Painter: DataWindow option.
NilIsNull	Whether to set the data value of the DropDownDataWindow to NULL when the user leaves the edit box blank. Values are: Yes — Make the Empty string NULL. No — Do not make the empty string NULL. Painter: Empty String is NULL option.

Property for dddw	Value
PercentWidth	An integer specifying the width of the drop-down portion of the DropDownDataWindow as a percentage of the column's width. Call GetChild *after* setting dddw.PercentWidth to get a valid reference to the column's DropDownDataWindow. Painter: Width of DropDown option.
Required	Whether the column is required. Values are: Yes — Required. No — (Default) Not required. Painter: Required option.
ShowList	Whether the ListBox portion of the DropDownDataWindow displays when the column has focus. A down arrow does not display at the right end of the DropDownDataWindow when dddw.ShowList is yes. Values are: Yes —Display the list whenever the column has the focus. No — Do not display the list until the user selects the column. Painter: Always Show List option.
UseAsBorder	Whether a down arrow displays at the right end of the DropDownDataWindow. Values are: Yes — Display the arrow. No — Do not display the arrow. Note that if ShowList is set to Yes, the column ignores the UseAsBorder property and the arrow never displays. Painter: Always Show Arrow option.
VScrollBar	Whether a vertical scroll bar displays in the DropDownDataWindow for long lists. Values are: Yes — Display a vertical scroll bar. No — Do not display a vertical scroll bar. Painter: V ScrollBar option.

Usage

DropDownDataWindows and GetChild When you set some of the dddw properties, as noted in the table, references to the DropDownDataWindow become invalid. Call GetChild again after changing these properties to obtain a valid reference.

To retrieve a DropDownDataWindow when the AutoRetrieve property is set to No, you can access the object data as follows:

```
DataWindowChild mgr_id
dw_1.GetChild ("dept_head_id", mgr_id)
```

```
mgr_id.SetTransObject (SQLCA)
mgr_id.Retrieve ( )
```

You can also pass a retrieval argument for the retrieve on the child DataWindow object.

Doing a reset to clear the data

When a DropDownDataWindow is retrieved, its data is kept with its own Data Object. If you retrieve the DropDownDataWindow and then set the AutoRetrieve property on the parent to No, the data for the child is not cleared on a reset and re-retrieve of the parent. To clear data from a DropDownDataWindow, you must call Reset on the child DataWindow object:

```
dw_1.GetChild ("dept_head_id", mgr_id)
mgr_id.reset ( )
```

In the painter Select the control and set values in the Properties view, Edit tab, when Style Type is DropDownDW.

Examples

```
string ls_data
ls_data = dw_1.Object.emp_status.dddw.AllowEdit")

dw_1.Object.emp_status.dddw.Case = "Any"

ls_data = dw_1.Describe("emp_status.dddw.AllowEdit")

dw_1.Modify("emp_status.dddw.Case='Any'")

dw_1.Modify("emp_status.dddw.DataColumn='status_id'")

dw_1.Modify("emp_status.dddw.Limit=30")

dw_1.Modify("emp_status.dddw.Name='d_status'")

dw_1.Modify("emp_status.dddw.PercentWidth=120")
```

ddlb.*property*

Description

Properties that control the appearance and behavior of a column with the DropDownListBox edit style.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Column controls

Syntax

PocketBuilder dot notation:

dw_control.Object.*columnname*.ddlb.*property*

Describe and Modify argument:

"*columnname*.ddlb.*property* { = *value* }"

Parameter	Description
columnname	The name of a column that has the DropDownListBox edit style.
property	A property for the DropDownListBox column. Properties and their settings are listed in the table below.
value	The value to be assigned to the property. For ddlb properties, value cannot be a DataWindow expression.

Property for ddlb	Value
AllowEdit	Whether the user can type a value as well as choose from the DropDownListBox's list. Values are: Yes — Typing is allowed. No — (Default) Typing is not allowed. Painter: Allow Editing option.
AutoHScroll	Whether the DropDownListBox automatically scrolls horizontally when the user enters or deletes data. Values are: Yes — (Default) Scroll horizontally automatically. No — Do not scroll automatically. Painter: Auto Horz Scroll option.
Case	The case of the text in the DropDownListBox. Values are: Any — Character of any case allowed. Upper — Characters converted to uppercase. Lower — Characters converted to lowercase. Painter: Case option.
Limit	An integer from 0–32767 specifying the maximum number of characters that can be entered in the DropDownListBox. Zero means unlimited. Painter: Limit option.
NilIsNull	Whether to set the data value of the DropDownListBox to NULL when the user leaves the edit box blank. Values are: Yes — Make the empty string NULL. No — Do not make the empty string NULL. Painter: Empty string is NULL option.

Property for ddlb	Value
Required	Whether the column is required. Values are: Yes — Required. No — (Default) Not required. Painter: Required option.
ShowList	Whether the ListBox portion of the DropDownListBox displays when the column has focus. A down arrow does not display at the right end of the DropDownListBox when ddlb.ShowList is yes. Values are: Yes — Display the list whenever the column has focus. No — Do not display the list until the user selects the column. Painter: Always Show List option.
Sorted	Whether the list in the DropDownListBox is sorted. Values are: Yes — The list is sorted. No — The list is not sorted. Painter: Sorted option.
UseAsBorder	Whether a down arrow displays at the right end of the DropDownListBox. Values are: Yes — Display the arrow. No — Do not display the arrow. Note that if ShowList is set to Yes, the column ignores the UseAsBorder property and the arrow never displays. Painter: Always Show Arrow option.
VScrollBar	Whether a vertical scroll bar displays in the DropDownListBox for long lists. Values are: Yes — Display a vertical scroll bar. No — Do not display a vertical scroll bar. Painter: V ScrollBar option.

Usage

In the painter Select the control and set the value in the Properties view, Edit tab, when Style Type is DropDownListBox.

Examples

```
string ls_data
ls_data = dw_1.Object.emp_status.ddlb.AllowEdit

dw_1.Object.emp_status.ddlb.Case = "Any"

ls_data = dw_1.Describe("emp_status.ddlb.AllowEdit")

dw_1.Modify("emp_status.ddlb.Case='Any'")

dw_1.Modify("emp_status.ddlb.Limit=30")
```

DefaultPicture

Description

Specifies whether a button displays a default picture for the button's action.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Button controls

Syntax

PocketBuilder dot notation:

dw_control.Object.*buttonname.*DefaultPicture

Describe and Modify argument:

"*buttonname.*DefaultPicture { = ' *value* ' }"

Parameter	Description
buttonname	The name of the button to which you want to assign an action.
value	Whether the action's default picture is used. Values are: Yes — Use the default picture. No — Do not use the default picture.

Usage

In the painter Select the control and set the value in the Properties view, General tab, Action Default Picture option. When the check box is not selected, you can specify a picture file name in the Picture File option. Button pictures can be BMP, GIF, or JPEG files.

Examples

```
dw_1.Object.b_name.DefaultPicture = "Yes"

setting = dw_1.Describe("b_name.DefaultPicture")

dw_1.Modify("b_name.DefaultPicture ='No'")
```

See also

HTMLGen.property
DefaultPicture
Filename

Depth

Description — The depth of a 3D graph.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to — Graph controls

Syntax — PocketBuilder dot notation:

dw_control.Object.*graphname.*Depth

Describe and Modify argument:

"*graphname.*Depth { = ' *depthpercent* ' }"

Parameter	Description
graphname	The graph control within the DataWindow for which you want to set the depth.
depthpercent	(*exp*) An integer whose value is the depth of the graph, specified as a percentage of the graph's width. *Depthpercent* can be a quoted DataWindow expression.

Usage — **In the painter** Select the control and set the value in the Properties view, General tab, Depth slider.

Examples

```
string setting
setting = dw_1.Object.graph_1.Depth

dw_1.Object.graph_1.Depth = 70

setting = dw_1.Describe("graph_1.Depth")

dw_1.Modify("graph_1.Depth='70'")
```

Detail_Bottom_Margin

Description — The size of the bottom margin of the DataWindow's detail area.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to — Style keywords

Syntax

SyntaxFromSQL:

Style (Detail_Bottom_Margin = *value*)

Parameter	Description
value	An integer specifying the size of the bottom margin of the detail area in the units specified for the DataWindow.

Examples

```
SQLCA.SyntaxFromSQL(sqlstring, &
'Style(...Detail_Bottom_Margin = 25 ...)', &
errstring)
```

Detail_Top_Margin

Description

The size of the top margin of the DataWindow's detail area.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Style keywords

Syntax

SyntaxFromSQL:

Style (Detail_Top_Margin = *value*)

Parameter	Description
value	An integer specifying the size of the top margin of the detail area in the units specified for the DataWindow.

Examples

```
SQLCA.SyntaxFromSQL(sqlstring, &
'Style(...Detail_Top_Margin = 25 ...)', &
errstring)
```

Detail.*property*

See Bandname.property.

DispAttr.*fontproperty*

Description

Settings for the appearance of various text components of a graph.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Properties of Graph controls, as noted throughout this discussion

Syntax

PocketBuilder dot notation:

dw_control.Object.*graphname*.*property*.DispAttr.*fontproperty*

Describe and Modify argument:

"*graphname*.*property*.DispAttr.*fontproperty* { = *value* }"

Parameter	Description
graphname	The Graph control in a DataWindow for which you want to get or set font appearance values.
property	A text component of the graph, such as an *Axis* keyword (Category, Series, or Values), Legend, Pie, or Title, specifying the graph component whose appearance you want to get or set. These properties have their own entries. These values are listed in the following table. You can also set font properties for the label of an axis with the following syntax: "*graphname.axis*.LabelDispAttr.*fontproperty* { = *value* }"
fontproperty	A property that controls the appearance of text in the graph. Properties and their settings are listed in the table below.
value	The value to be assigned to *fontproperty*. *Value* can be a quoted DataWindow expression.

Property for DispAttr	Value
Alignment	(*exp*) The alignment of the text. Values are: 0 — Left 1 — Right 2 — Center Painter: Alignment option.

Property for DispAttr	Value
AutoSize	(*exp*) Whether the text element should be autosized according to the amount of text being displayed. Values are: 0 — Do not autosize 1 — Autosize Painter: Autosize check box.
BackColor	(*exp*) A long value specifying the background color of the text. Painter: BackColor option.
DisplayExpression	An expression whose value is the label for the graph component. The default expression is the property containing the text for the graph component. The expression can include the text property and add other variable text. Painter: Display Expression option.
Font.CharSet	(*exp*) An integer specifying the character set to be used. Values are: 0 — ANSI 1 — The default character set for the specified font 2 — Symbol 128 — Shift JIS 255 — OEM Painter: FontCharSet option.
Font.Escapement	(*exp*) An integer specifying the rotation for the baseline of the text in tenths of a degree. For example, a value of 450 rotates the text 45 degrees. 0 is horizontal. Painter: Escapement option.
Font.Face	(*exp*) A string specifying the name of the font face, such as Arial or Courier. Painter: FaceName option.
Font.Family	(*exp*) An integer specifying the font family (Windows uses both face and family to determine which font to use). Values are: 0 — AnyFont 1 — Roman 2 — Swiss 3 — Modern 4 — Script 5 — Decorative Painter: Family option.

Property for DispAttr	Value
Font.Height	(*exp*) An integer specifying the height of the text in the unit measure for the DataWindow. To specify size in points, specify a negative number. Painter: Size option, specified in points (not available when AutoSize is checked).
Font.Italic	(*exp*) Whether the text should be italic. Values are: 0 — Not italic (default) 1 — Italic Painter: Italic option.
Font.Orientation	Same as Escapement.
Font.Pitch	(*exp*) The pitch of the font. Values are: 0 — The default pitch for your system 1 — Fixed 2 — Variable Painter: Pitch option.
Font.Strikethrough	(*exp*) Whether the text should be crossed out. Values are: 0 — Not crossed out (default) 1 — Crossed out Painter: Strikeout option.
Font.Underline	(*exp*) Whether the text should be underlined. Values are: 0 — Not underlined (default) 1 — Underlined Painter: Underline option.
Font.Weight	(*exp*) An integer specifying the weight of the text—for example, 400 for normal or 700 for bold. Painter: Set indirectly using the Bold option.
Font.Width	(*exp*) An integer specifying the width of the font in the unit of measure specified for the DataWindow. Width is usually unspecified, which results in a default width based on the other properties. Painter: Width option.
Format	(*exp*) A string containing the display format for the text. Painter: Format option.
TextColor	(*exp*) A long specifying the color to be used for the text. Painter: TextColor option.

Usage

In the painter Select the control and set values in the Properties view, Text tab. Settings apply to the selected item in the Text Object list box.

Examples

```
setting = &
dw_1.Object.Category.LabelDispAttr.Font.Face

dw_1.Object.Category.LabelDispAttr.Font.Face = "Arial"

setting = &
dw_1.Describe("Category.LabelDispAttr.Font.Face")

dw_1.Modify("Category.LabelDispAttr.Font.Face='Arial'"
)

dw_1.Modify("Title.DispAttr.DisplayExpression=" &
+ "'Title + ~"~n~" + Today()'")
```

DisplayType

Description

The way the OLE Object control displays the OLE object it contains. It can display an icon or an image of the object's contents. The image is reduced to fit inside the OLE container.

PocketBuilder	✗
PowerBuilder	✓

Both the icon and the image are provided by the OLE server. If the OLE server does not support a contents view, PowerBuilder displays an icon even if DisplayType is set to contents.

Applies to

OLE Object controls

Syntax

PowerBuilder dot notation:

dw_control.Object.*olecontrolname*.DisplayType

Describe and Modify argument:

"*olecontrolname*.DisplayType { = ' *type* ' }"

Edit.*property*

Description

Settings that affect the appearance and behavior of columns whose edit style is Edit.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Column controls

Syntax

PocketBuilder dot notation:

dw_control.Object.*columnname*.Edit.*property*

Describe and Modify argument:

"*columnname*.Edit.*property* { = *value* }"

SyntaxFromSQL:

Column (Edit.*property* = *value*)

Parameter	Description
columnname	The column with the Edit edit style for which you want to get or set property values. You can specify the column name or a pound sign (#) and the column number.
property	A property for the column's Edit style. Properties and their settings are listed in the table below. The table identifies the properties you can use with SyntaxFromSQL.
value	The value to be assigned to the property. For most Edit properties, you cannot specify a DataWindow expression. The exception is Edit.Format.

Property for Edit	Value
AutoHScroll	Whether the edit control scrolls horizontally automatically when data is entered or deleted. Values are: Yes — Scroll horizontally automatically. No — Do not scroll horizontally automatically. You can use AutoHScroll with SyntaxFromSQL. The setting applies to all the columns in the generated syntax. Painter: Auto Horz Scroll option.
AutoRetrieve	Whether an embedded DropDownDataWindow is retrieved automatically at the same time as the retrieve on the parent DataWindow. Values are: Yes — (Default) Child DataWindow automatically retrieved. No — Child DataWindow must be retrieved programmatically. A subsequent reset or retrieval on the parent does not reset the contents of the child DataWindow. Painter: AutoRetrieve option for DropDownDataWindow controls.

Property for Edit	Value
AutoSelect	Whether to select the contents of the edit control automatically when it receives focus. Values are: Yes — Select automatically. No — Do not select automatically. You can use AutoSelect with SyntaxFromSQL. The setting applies to all the columns in the generated syntax. Painter: Auto Selection option.
AutoVScroll	Whether the edit box scrolls vertically automatically when data is entered or deleted. Values are: Yes — Scroll vertically automatically. No — Do not scroll vertically automatically. You can use AutoVScroll with SyntaxFromSQL. The setting applies to all the columns in the generated syntax. Painter: Auto Vert Scroll option.
Case	The case of the text in the edit control. Values are: Any — Character of any case allowed. Upper — Characters converted to uppercase. Lower — Characters converted to lowercase. Painter: Case option.
CodeTable	Whether the column has a code table. Values are: Yes — Code table defined. No — No code table defined. Painter: Use Code Table option.
DisplayOnly	Whether the column is display only. Values are: Yes — Do not allow the user to enter data; make the column display only. No — (Default) Allow the user to enter data. Painter: Display Only option. For conditional control over column editing, use the Protect property.
FocusRectangle	Whether a dotted rectangle (the focus rectangle) will surround the current row of the column when the column has focus. Values are: Yes — (Default) Display the focus rectangle. No — Do not display the focus rectangle. You can use FocusRectangle with SyntaxFromSQL. The setting applies to all the columns in the generated syntax. Painter: Show Focus Rectangle option.

Property for Edit	Value
Format	(*exp*) A string containing the display format of the edit control. The value for Format is quoted and can be a DataWindow expression. Painter: Format option (do not use quotes around the value).
HScrollBar	Whether a horizontal scroll bar displays in the edit control. Values are: Yes — Display the horizontal scroll bar. No — Do not display the horizontal scroll bar. Painter: Horz Scroll Bar option.
InputEditMode	When an editable column has focus, the SIP type on a Pocket PC or the input method edit mode on a Smartphone device. Values are: 0 — (Default) Does not change the current SIP type on a Pocket PC or the current edit mode on a Smartphone 1 — Numeric mode for a Smartphone 2 — T9 mode for a Smartphone 3 — Multipress lowercase mode for a Smartphone 4 — T9 uppercase mode for a Smartphone 5 — T9 first letter uppercase for a Smartphone 6 — Multipress uppercase mode for a Smartphone 7 — Multipress first letter uppercase for a Smartphone 11 — SIP Keyboard mode for a Pocket PC 12 — SIP Jot mode for a Pocket PC 13 — SIP Block mode for a Pocket PC 14 — SIP WordLogic mode for a Pocket PC 15 — SIP Transcriber mode for a Pocket PC (Using the Transcriber mode for a DataWindow text field leads to unpredictable results.) 16 — Fitaly SIP keyboard for a Pocket PC
Limit	A number specifying the maximum number of characters (0 to 32,767) that the user can enter. 0 means unlimited. Painter: Limit option.
Name	A string whose value is the name of the predefined edit style associated with the column. Named styles are defined in the Database painter and can be reused. Specifying a name that has not been previously defined associates the name with the column but does not define a new edit style. Painter: Style Name option.

Property for Edit	Value
NilIsNull	Whether to set the value of the edit control to NULL when the user leaves it blank. Values are: Yes — Make the Empty string NULL. No — Do not make the empty string NULL. Painter: Empty String is NULL option.
Password	Whether to assign secure display mode to the column. When the user enters characters, they display as asterisks (*) Values are: Yes — Assign secure display mode to the column. No — Do not assign secure-display mode to the column. If you change the Password property, you should also change the Format property to display the results you want (for example, *****). Painter: Password option.
Required	Whether the column is required. Values are: Yes — It is required. No — It is not required. Painter: Required option.
SipOnFocus	Whether to display or minimize the SIP when the column receives focus. Values are: Yes — SIP opens automatically. No — SIP closes automatically. Painter: Show SIP on Focus option.
Style	(*Describe only*) Returns the edit style of the column. Painter: Style Type option.
ValidateCode	Whether the code table will be used to validate user-entered values. Values are: Yes — Use the code table. No — Do not use the code table. Painter: Validate option, available when Use Code Table is selected.
VScrollBar	Whether a vertical scroll bar displays in the line edit. Values are: Yes — Display vertical scroll bars. No — Do not display vertical scroll bars. Painter: Vert Scroll Bar option.

Usage

In the painter Select the control and set values in the Properties view, Edit tab, when Style Type is Edit.

Examples

```
string setting
setting = dw_1.Object.emp_name.Edit.AutoHScroll

dw_1.Object.emp_name.Edit.Required = "no"

setting = dw_1.Describe("emp_name.Edit.AutoHScroll")

dw_1.Modify("emp_name.Edit.Required=no")
```

EditMask.*property*

Description

Settings that affect the appearance and behavior of columns with the EditMask edit style.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Column controls

Syntax

PocketBuilder dot notation:

dw_control.Object.*columnname*.EditMask.*property*

Describe and Modify argument:

"*columnname*.EditMask.*property* { = *value* }"

Parameter	Description
columnname	The column with the EditMask edit style for which you want to get or set property values. You can specify the column name or a pound sign (#) and the column number.
property	A property for the column's EditMask style. Properties and their settings are listed in the table below.
value	The value to be assigned to the property. For EditMask properties, you cannot specify a DataWindow expression.

Property for EditMask	Value
AutoSkip	Whether the EditMask will automatically skip to the next field when the maximum number of characters has been entered: Yes — Skip automatically. No — Do not skip automatically. Painter: AutoSkip option.

Property for EditMask	Value
CodeTable	Whether the column has a code table. Values are: Yes — Code table defined. No — No code table defined. Painter: Code Table option. When selected, Display Value and Data Value are displayed for specifying code table entries.
FocusRectangle	Whether a dotted rectangle (the focus rectangle) will surround the current row of the column when the column has focus. Values are: Yes — (Default) Display the focus rectangle. No — Do not display the focus rectangle. Painter: Show Focus Rectangle option.
InputEditMode	When an editable column has focus, the SIP type on a Pocket PC or the input method edit mode on a Smartphone device. Values are: 0 — (Default) Does not change the current SIP type on a Pocket PC or the current edit mode on a Smartphone 1 — Numeric mode for a Smartphone 2 — T9 mode for a Smartphone 3 — Multipress lowercase mode for a Smartphone 4 — T9 uppercase mode for a Smartphone 5 — T9 first letter uppercase for a Smartphone 6 — Multipress uppercase mode for a Smartphone 7 — Multipress first letter uppercase for a Smartphone 11 — SIP Keyboard mode for a Pocket PC 12 — SIP Jot mode for a Pocket PC 13 — SIP Block mode for a Pocket PC 14 — SIP WordLogic mode for a Pocket PC 15 — SIP Transcriber mode for a Pocket PC (Using the Transcriber mode for a DataWindow text field leads to unpredictable results.) 16 — Fitaly SIP keyboard for a Pocket PC
Mask	A string containing the edit mask for the column. Painter: Mask option.
ReadOnly	Whether the column is read-only. This property is valid only if EditMask.Spin is set to Yes. Values are: Yes — Do not allow the user to enter data; make the column read-only. No — (Default) Allow the user to enter data. Painter: Read Only option.

Property for EditMask	Value
Required	Whether the column is required. Values are: Yes — It is required. No — It is not required. Painter: Required option.
SipOnFocus	Whether to display or minimize the SIP when the column receives focus. Values are: Yes — SIP opens automatically. No — SIP closes automatically. Painter: Show SIP on Focus option.
Spin	Whether the user can scroll through a list of possible values for the column with a spin control. Values are: Yes — Display a spin control. No — (Default) Do not display a spin control. Painter: Spin Control option.
SpinIncr	An integer indicating the amount to increment the spin control's values. The default for numeric values is 1; for dates, 1 year; and for time, 1 minute. For columns that are not numeric, date, or time, the spin control scrolls through values in an associated code table. If the EditMask.CodeTable property is No, the spin increment has no effect for these columns. Painter: Spin Increment option (available for numeric, date, and time columns).
SpinRange	A string containing the maximum and minimum values for the column that will display in the spin control. The two values are separated by a tilde (~). This property is effective only if EditMaskSpin is Yes. Because the SpinRange string is within another quoted string, the tilde separator becomes four tildes in PocketBuilder, which reduces to a single tilde when parsed. The format for the string is: "EditMask.SpinRange = ' *minval*~~~~*maxval* ' " Painter: Spin Range group, Spin Min and Spin Max options (available for numeric, date, and time columns).

Property for EditMask	Value
UseFormat	Whether a Format Display mask is used for a column's display. A Format Display mask is used only when the column does not have focus. Values are: Yes — Use a Format Display mask. No — (Default) Do not use a Format Display mask. Painter: Use Format option.

Usage

In the painter Select the control and set values in the Properties view, Edit tab, when Style is EditMask.

Examples

```
string setting
setting = dw_1.Object.emp_status.EditMask.Spin

dw_1.Object.emp_bonus.EditMask.SpinIncr = 1000

dw_1.Object.id.EditMask.SpinRange = "0~~~~10"

setting = dw_1.Describe("emp_status.EditMask.Spin")

dw_1.Modify("emp_bonus.EditMask.SpinIncr=1000")

dw_1.Modify("emp_bonus.EditMask.SpinRange='0~~~~5000'"
)
```

Elevation

Description

The elevation in a 3D graph.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Graph controls

Syntax

PocketBuilder dot notation:

dw_control.Object.*graphname*.Elevation

Describe and Modify argument:

"*graphname*.Elevation { = ' *integer* ' }"

Parameter	Description
graphname	The name of the graph control in the DataWindow for which you want to get or set the elevation.

Parameter	Description
integer	(*exp*) An integer specifying the elevation of the graph. Elevation can be a quoted DataWindow expression.

Usage

In the painter Select the control and set the value in the Properties view, General tab, Elevation scroll bar (enabled when a 3D graph type is selected).

Examples

```
string setting
setting = dw_1.Object.graph_1.Elevation

dw_1.Object.graph_1.Elevation = 35

setting = dw_1.Describe("graph_1.Elevation")

dw_1.Modify("graph_1.Elevation=35")

dw_1.Modify("graph_1.Elevation='10~tIf(...,20,30)'")
```

EllipseHeight

Description

The radius of the vertical part of the corners of a RoundRectangle.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

RoundRectangle controls

Syntax

PocketBuilder dot notation:

dw_control.Object.*rrectname*.EllipseHeight

Describe and Modify argument:

"*rrectname*.EllipseHeight { = ' *integer* ' }"

Parameter	Description
rrectname	The name of the RoundRectangle control in the DataWindow for which you want to get or set the ellipse height.
integer	(*exp*) An integer specifying the radius of the vertical part of the corners of a RoundRectangle in the DataWindow's unit of measure. EllipseHeight can be a quoted DataWindow expression.

Usage

In the painter Select the control and set the value in the Properties view, General tab.

Examples

```
string setting
setting = dw_1.Object.rrect_1.EllipseHeight

dw_1.Object.rrect_1.EllipseHeight = 35

setting = dw_1.Describe("rrect_1.EllipseHeight")

dw_1.Modify("rrect_1.EllipseHeight=35")

dw_1.Modify("rrect_1.EllipseHeight='10~tIf(...,20,30)'
")
```

EllipseWidth

Description

The radius of the horizontal part of the corners of a RoundRectangle.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

RoundRectangle controls

Syntax

PocketBuilder dot notation:

dw_control.Object.*rrectname*.EllipseWidth

Describe and Modify argument:

"*rrectname*.EllipseWidth { = ' *integer* ' }"

Parameter	Description
rrectname	The name of the RoundRectangle control in the DataWindow for which you want to get or set the ellipse width.
integer	(*exp*) An integer specifying the radius of the horizontal part of the corners of a RoundRectangle in the DataWindow's unit of measure. EllipseWidth can be a quoted DataWindow expression.

Usage

In the painter Select the control and set the value in the Properties view, General tab.

Examples

```
string setting
setting = dw_1.Object.rrect_1.EllipseWidth

dw_1.Object.rrect_1.EllipseWidth = 35

setting = dw_1.Describe("rrect_1.EllipseWidth")
```

```
dw_1.Modify("rrect_1.EllipseWidth=35")

dw_1.Modify("rrect_1.EllipseWidth='10~tIf(...,20,30)'"
)
```

Enabled

Description

Determines whether a button control in a DataWindow is enabled.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Button controls

Syntax

PocketBuilder dot notation:

dw_control.Object.*buttonname.*Enabled

Describe and Modify argument:

"*buttonname.*Enabled { = ' *value* ' }"

Parameter	Description
buttonname	The name of the button that you want to enable or disable.
value	Whether the button is enabled. Values are: Yes — (Default) The button is enabled. No — The button is disabled.

Usage

In the painter Select the control and set the value in the Properties view, General tab, Enabled option.

When the Enabled check box is cleared, or the Enabled property is otherwise set to false, the button control is grayed and its actions are not performed.

Examples

```
dw_1.Object.b_name.Enabled = "No"

setting = dw_1.Describe("b_name.Enabled")

dw_1.Modify("b_name.Enabled ='No'")
```

Export.PDF.Distill.CustomPostScript

Description

Setting that enables you to specify the PostScript printer driver settings used when data is exported to PDF using the Distill! method.

PocketBuilder	✗
PowerBuilder	✓

Applies to

DataWindow objects

Syntax

PowerBuilder dot notation:

dw_control.Object.DataWindow.Export.PDF.Distill.CustomPostScript

Describe and Modify argument:

"DataWindow.Export.PDF.Distill.CustomPostScript { = '*value* ' }"

Parameter	Description
value	(*exp*) Whether the printer specified in the DataWindow.Printer property is used when data is exported to PDF. Values are: • Yes — The printer specified in DataWindow.Printer is used for PDF export. • No — The default printer is used for PDF export (default).

Export.PDF.Method

Description

Setting that determines whether data is exported to PDF from a DataWindow object by printing to a PostScript file and distilling to PDF, or by saving in XSL Formatting Objects (XSL-FO) format and processing to PDF.

PocketBuilder	✗
PowerBuilder	✓

Applies to

DataWindow objects

Syntax

PowerBuilder dot notation:

dw_control.Object.DataWindow.Export.PDF.Method

Describe and Modify argument:

"DataWindow.Export.PDF.Method { = '*value* ' }"

Export.PDF.XSLFOP.Print

Description

Setting that enables you to send a DataWindow object directly to a printer using platform-independent Java printing when using the XSL-FO method to export to PDF. This is an option of the Apache FOP processor.

PocketBuilder	✗
PowerBuilder	✓

Applies to

DataWindow objects

Syntax

PowerBuilder dot notation:

dw_control.Object.DataWindow.PDF.XSLFOP.Print

Describe argument:

"DataWindow.PDF.XSLFOP.Print { = '*value* ' }"

Export.XML.HeadGroups

Description

Setting that causes elements, attributes, and all other items above the Detail Start element in an XML export template for a group DataWindow to be iterated for each group in the exported XML.

PocketBuilder	✗
PowerBuilder	✓

Applies to

DataWindow objects

Syntax

PowerBuilder dot notation:

dw_control.Object.DataWindow.Export.XML.HeadGroups

Describe and Modify argument:

"DataWindow.Export.XML.HeadGroups { = '*value* ' }"

Export.XML.IncludeWhitespace

Description

Setting that determines whether the XML document is formatted by inserting whitespace characters (carriage returns, linefeeds, tabs, and spacebar spaces).

PocketBuilder	✗
PowerBuilder	✓

Applies to DataWindow objects

Syntax PowerBuilder dot notation:

dw_control.Object.DataWindow.Export.XML.IncludeWhitespace

Describe and Modify argument:

"DataWindow.Export.XML.IncludeWhitespace { = '*value* ' }"

Export.XML.MetaDataType

Description Setting that controls the type of metadata generated with the XML exported from a DataWindow object using the SaveAs method or a .Data.XML expression.

PocketBuilder	✗
PowerBuilder	✓

Applies to DataWindow objects

Syntax PowerBuilder dot notation:

dw_control.Object.DataWindow.Export.XML.MetaDataType

Describe and Modify argument:

"DataWindow.Export.XML.MetaDataType { = '*value* ' }"

Export.XML.SaveMetaData

Description Setting that controls the storage format for the metadata generated with the XML exported from a DataWindow object using the SaveAs method or a .Data.XML expression.

PocketBuilder	✗
PowerBuilder	✓

Applies to DataWindow objects

Syntax PowerBuilder dot notation:

dw_control.Object.DataWindow.Export.XML.SaveMetaData

Describe and Modify argument:

"DataWindow.Export.XML.SaveMetaData { = '*value* ' }"

Export.XML.TemplateCount

Description

The number of XML export templates associated with a DataWindow object.

PocketBuilder	✗
PowerBuilder	✓

Applies to

DataWindow objects

Syntax

PowerBuilder dot notation:

dw_control.Object.DataWindow.Export.XML.TemplateCount

Describe argument:

"DataWindow.Export.XML.TemplateCount"

Export.XML.Template[ ].Name

Description

The name of an XML export template associated with a DataWindow object.

PocketBuilder	✗
PowerBuilder	✓

Applies to

DataWindow objects

Syntax

PowerBuilder dot notation:

dw_control.Object.DataWindow.Export.XML.Template[*num*].Name

Describe argument:

"DataWindow.Export.XML.Template[*num*]Name"

Export.XML.UseTemplate

Description

Setting that optionally controls the logical structure of the XML exported from a DataWindow object using the SaveAs method or the .Data.XML property.

PocketBuilder	✗
PowerBuilder	✓

Applies to

DataWindow objects

Syntax

PowerBuilder dot notation:

dw_control.Object.DataWindow.Export.XML.UseTemplate

Describe and Modify argument:

"DataWindow.Export.XML.UseTemplate { = '*value* ' }"

Expression

Description

The expression for a computed field control in the DataWindow. The expression is made up of calculations and DataWindow expression functions. The DataWindow evaluates the expression to get the value it will display in the computed field.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Computed field controls

Syntax

PocketBuilder dot notation:

dw_control.Object.*computename*.Expression

Describe and Modify argument:

"*computename*.Expression { = '*string* ' }"

Parameter	Description
computename	The name of the computed field control in the DataWindow for which you want to get or set the expression.
string	A string whose value is the expression for the computed field.

Usage

In the painter Select the control and set the value in the Properties view, General tab, Expression option. The More button displays the Modify Expression dialog, which provides help in specifying the expression. The Verify button tests the expression.

Examples

```
setting = dw_1.Object.comp_1.Expression

dw_1.Object.comp_1.Expression = "avg(salary for all)"

setting = dw_1.Describe("comp_1.Expression")

dw_1.Modify("comp_1.Expression='avg(salary for all)'")
```

Filename

Description

The file name containing the image for a Picture or Button control in the DataWindow. If no image is specified for a Button control, only text is used for the button label.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Picture controls

Syntax

PocketBuilder dot notation:

dw_control.Object.*controlname*.Filename

Describe and Modify argument:

"*controlname*.Filename { = ' *filestring* ' }"

Parameter	Description
controlname	The name of the Picture or Button control in the DataWindow for which you want to get or set the image file name.
filestring	(*exp*) A string containing the name of the file that contains the image. *Filestring* can be a quoted DataWindow expression. Button pictures can be BMP, GIF, or JPEG files. You can use a URL instead of a full path name, and if you set the HTMLGen.ResourceBase property to the URL address, you only need to specify a relative file name for this string. If you include the name of the file containing the image in the executable for the application, PocketBuilder will always use that image; you cannot use Modify to change the image.

Usage

In the painter For a Picture control, select the control and set the value in the Properties view, General tab, File Name option. For a Button control, select the control and set the value in the Properties view, General tab, Picture File option. The Action Default Picture check box must be cleared to set the value for the picture file.

Examples

Example for a Picture control:

```
setting = dw_1.Object.bitmap_1.Filename
dw_1.Object.bitmap_1.Filename = "exclaim.bmp"
setting = dw_1.Describe("bitmap_1.Filename")
dw_1.Modify("bitmap_1.Filename='exclaim.bmp'")
```

Example for a Button control:

```
dw_1.Object.b_name.FileName = "logo.gif"
ls_data = dw_1.Describe("b_name.FileName")
dw_1.Modify("b_name.FileName = 'logo.jpg'")
```

See also DefaultPicture

FirstRowOnPage

Description The first row currently visible in the DataWindow.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to DataWindows

Syntax PocketBuilder dot notation:

dw_control.Object.DataWindow.FirstRowOnPage

Describe argument:

"DataWindow.FirstRowOnPage"

Examples

```
string setting
setting = dw_1.Object.DataWindow.FirstRowOnPage
setting = dw_1.Describe("DataWindow.FirstRowOnPage")
```

Font.Bias

Description The way fonts are manipulated in the DataWindow during execution.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to DataWindows

Syntax PocketBuilder dot notation:

dw_control.Object.DataWindow.Font.Bias

Describe and Modify argument:

"DataWindow.Font.Bias { = *biasvalue* }"

Parameter	Description
biasvalue	An integer indicating how the fonts will be manipulated at execution. *Biasvalue* cannot be a DataWindow expression. Values are: 0 — As display fonts 1 — As printer fonts 2 — Neutral; no manipulation will take place

Examples

```
string setting
setting = dw_1.Object.DataWindow.Font.Bias

dw_1.Object.DataWindow.Font.Bias = 1

setting = dw_1.Describe("DataWindow.Font.Bias")

dw_1.Modify("DataWindow.Font.Bias=1")
```

Font.*property*

Description

Settings that control the appearance of fonts within a DataWindow, except for graphs, which have their own settings (see DispAttr).

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Button, Column, Computed Field, GroupBox, and Text controls

Syntax

PocketBuilder dot notation:

dw_control.Object.*controlname*.Font.*property*

Describe and Modify argument:

"*controlname*.Font.*property* { = ' *value* ' }"

SyntaxFromSQL:

Column(Font.*property* = *value*)

Text(Font.*property* = *value*)

Parameter	Description
controlname	The name of a column, computed field, or text control for which you want to get or set font properties. For a column, you can specify its name or a pound sign (#) followed by the column number. When generating DataWindow syntax with SyntaxFromSQL, the Font settings apply to all columns or all text controls.
property	A property of the text. The properties and their values are listed in the table below.
value	The value to be assigned to the property. *Value* can be a quoted DataWindow expression.

Property for Font	Value
CharSet	(*exp*) An integer specifying the character set to be used. Values are: 0 — ANSI 1 — The default character set for the specified font 2 — Symbol 128 — Shift JIS 255 — OEM Painter: Font tab, CharSet option.
Escapement	(*exp*) An integer specifying the rotation for the baseline of the text in tenths of a degree. For example, a value of 450 rotates the text 45 degrees. 0 is horizontal. Painter: Font tab, Escapement option.
Face	(*exp*) A string specifying the name of the font face, such as Arial or Courier. Painter: Font tab, FaceName option or StyleBar.
Family	(*exp*) An integer specifying the font family (Windows uses both face and family to determine which font to use). Values are: 0 — AnyFont 1 — Roman 2 — Swiss 3 — Modern 4 — Script 5 — Decorative Painter: Font tab, Family option.

Property for Font	Value
Height	(*exp*) An integer specifying the height of the text in the unit measure for the DataWindow. To specify size in points, specify a negative number. Painter: Font tab, Size option (specified in points) or StyleBar or Expressions tab.
Italic	(*exp*) Whether the text should be italic. The default is no. Painter: Font tab, Italic check box or StyleBar.
Pitch	(*exp*) The pitch of the font. Values are: 0 — The default pitch for your system 1 — Fixed 2 — Variable Painter: Font tab, Pitch option.
Strikethrough	(*exp*) Whether the text should be crossed out. The default is no. Painter: Font tab, Strikeout check box.
Underline	(*exp*) Whether the text should be underlined. The default is no. Painter: Font tab, Underline check box or StyleBar.
Weight	(*exp*) An integer specifying the weight of the text; for example, 400 for normal or 700 for bold. Painter: Set indirectly using Font tab, Bold check box or the StyleBar, Bold button.
Width	(*exp*) An integer specifying the average character width of the font in the unit of measure specified for the DataWindow. Width is usually unspecified, which results in a default width based on the other properties. Painter: Set indirectly using font selection.

Usage

In the painter Select the control and set the value using the:

- Properties view, Font tab
- For some font settings, StyleBar

Examples

```
dw_1.Object.emp_name_t.Font.Face

dw_1.Object.emp_name_t.Font.Face = "Arial"

dw_1.Describe("emp_name_t.Font.Face")

dw_1.Modify("emp_name_t.Font.Face='Arial'")
```

Footer.*property*

See Bandname.property.

Format

Description

The display format for a column.

You can use the GetFormat and SetFormat functions instead of Describe and Modify to get and change a column's display format. The advantage to using Modify is the ability to specify an expression.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Column and Computed Field controls

Syntax

PocketBuilder dot notation:

dw_control.Object.*controlname*.Format

Describe and Modify argument:

"*controlname*.Format { = ' *value* ' }"

Parameter	Description
controlname	The name of the column or computed field for which you want to get or set the display format.
value	(*exp*) A string specifying the display format. See the *Users Guide* for information on constructing display formats. *Value* can be a quoted DataWindow expression.

Usage

In the painter Select the control and set the value in the Properties view, Format tab.

Examples

```
setting = dw_1.Object.phone.Format

dw_1.Object."phone.Format = "[red](@@@)@@@-
@@@@;'None'"

setting = dw_1.Describe("phone.Format")

dw_1.Modify( &
"phone.Format='[red](@@@)@@@-@@@@;~~~'None~~~''")
```

See also

GetFormat function in the *PowerScript Reference*
SetFormat function in the *PowerScript Reference*

GraphType

Description

The type of graph, such as bar, pie, column, and so on.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Graph controls

Syntax

PocketBuilder dot notation:

dw_control.Object.*graphname*.GraphType

Describe and Modify argument:

"*graphname*.GraphType { = ' *typeinteger* ' }"

Parameter	Description
graphname	The graph control for which you want to get or change the type.
typeinteger	(*exp*) An integer identifying the type of graph in the DataWindow object. *Typeinteger* can be a quoted DataWindow expression. Values are: 1 — Area 2 — Bar 3 — Bar3D 4 — Bar3DObj 5 — BarStacked 6 — BarStacked3DObj 7 — Col 8 — Col3D 9 — Col3DObj 10 — ColStacked 11 — ColStacked3DObj 12 — Line 13 — Pie 14 — Scatter 15 — Area3D 16 — Line3D 17 — Pie3D

Usage

In the painter Select the control and set the value in the Properties view, General tab.

Examples

```
string setting
setting = dw_1.Object.graph_1.GraphType

dw_1.Object.graph_1.GraphType = 17
```

```
setting = dw_1.Describe("graph_1.GraphType")
dw_1.Modify("graph_1.GraphType=17")
```

Grid.ColumnMove

Description

Whether the user can rearrange columns by dragging.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

DataWindows

Syntax

PocketBuilder dot notation:

dw_control.Object.DataWindow.Grid.ColumnMove

Describe and Modify argument:

"DataWindow.Grid.ColumnMove { = *value* } "

Parameter	Description
value	Whether the user can rearrange columns. Values are: Yes — The user can drag columns. No — The user cannot drag columns.

Usage

In the painter Select the DataWindow object by deselecting all controls; then set the value in the Properties view, General tab, Grid group, Column Moving check box (available when the presentation style is Grid or Crosstab).

Examples

```
string setting
setting = dw_1.Object.DataWindow.Grid.ColumnMove
dw_1.Object.DataWindow.Grid.ColumnMove = No
setting = dw_1.Describe("DataWindow.Grid.ColumnMove")
dw_1.Modify("DataWindow.Grid.ColumnMove=No")
```

Grid.Lines

Description

The way grid lines display and print in a DataWindow whose presentation style is Grid or Crosstab.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

DataWindows

Syntax

PocketBuilder dot notation:

dw_control.Object.DataWindow.Grid.Lines

Describe and Modify argument:

"DataWindow.Grid.Lines { = *value* }"

Parameter	Description
value	An integer specifying whether grid lines are displayed on the screen and printed. Values are: 0 — Yes, grid lines are displayed and printed. 1 — No, grid lines are not displayed and printed. 2 — Grid lines are displayed, but not printed. 3 — Grid lines are printed, but not displayed.

Usage

In the painter Select the DataWindow object by deselecting all controls; then set the value in the Properties view, General tab, Grid group, Display option (available when the presentation style is Grid or Crosstab).

Examples

```
string setting
setting = dw_1.Object.DataWindow.Grid.Lines

dw_1.Object.DataWindow.Grid.Lines = 2

setting = dw_1.Describe("DataWindow.Grid.Lines")

dw_1.Modify("DataWindow.Grid.Lines=2")
```

GroupBy

Description

A comma-separated list of the columns or expressions that control the grouping of the data transferred from the DataWindow to the OLE object. When there is more than one grouping column, the first one is the primary group and the columns that follow are nested groups.

PocketBuilder	✗
PowerBuilder	✓

Applies to

OLE Object controls

Syntax

PowerBuilder dot notation:

dw_control.Object.*olecontrolname*.GroupBy

Describe and Modify argument:

"*olecontrolname*.GroupBy { = ' *columnlist* ' }"

Header_Bottom_Margin

Description

The size of the bottom margin of the DataWindow's header area. Header_Bottom_Margin is meaningful only when type is Grid or Tabular.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Style keywords

Syntax

SyntaxFromSQL:

Style (Header_Bottom_Margin = *value*)

Parameter	Description
value	An integer specifying the size of the bottom margin of the header area in the units specified for the DataWindow. The bottom margin is the distance between the bottom of the header area and the last line of the header.

Examples

```
SQLCA.SyntaxFromSQL(sqlstring, &
'Style(...Header_Bottom_Margin = 25 ...)', &
errstring)
```

Header_Top_Margin

Description

The size of the top margin of the DataWindow's header area. Header_Top_Margin is meaningful only when type is Grid or Tabular.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Style keywords

Syntax

SyntaxFromSQL:

Style (Header_Top_Margin = *value*)

Parameter	Description
value	An integer specifying the size of the top margin of the header area in the units specified for the DataWindow. The top margin is the distance between the top of the header area and the first line of the header.

Examples

```
SQLCA.SyntaxFromSQL(sqlstring, &
'Style(...Header_Top_Margin = 500 ...)', errstring)
```

Header.*property*

See Bandname.property.

Header.#.*property*

See Bandname.property.

Height

Description

The height of a control in the DataWindow.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Button, Column, Computed Field, Graph, GroupBox, OLE, Oval, Picture, Rectangle, Report, RoundRectangle, TableBlob, and Text controls

Syntax

PocketBuilder dot notation:

dw_control.Object.*controlname*.Height

Describe and Modify argument:

"*controlname*.Height { = ' *value* ' }"

Parameter	Description
controlname	The control within the DataWindow whose height you want to get or set.
value	(*exp*) An integer specifying the height of the control in the unit of measure specified for the DataWindow. *Value* can be a quoted DataWindow expression.

Usage

In the painter Select the control and set the value in the Properties view, Position tab.

Examples

```
string setting
setting = dw_1.Object.empname.Height

dw_1.Object.empname.Height = 50

setting = dw_1.Describe("empname.Height")

dw_1.Modify("empname.Height=50")
```

Height.AutoSize

Description

Whether the control's width should be held constant and its height adjusted so that all the data is visible. This property is for use with read-only controls and printed reports. It should not be used with data entry fields or controls.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Column, Computed Field, Report, and Text controls

Syntax

PocketBuilder dot notation:

dw_control.Object.*controlname*.Height.AutoSize

Describe and Modify argument:

"*controlname*.Height.AutoSize { = *value* }"

Parameter	Description
controlname	The control for which you want to get or set the AutoSize property.
value	Whether the width or height of the control will be adjusted to display all the data. The height is limited to what can fit on the page. Values are: No — Use the height defined in the painter. Yes — Calculate the height so that all the data is visible.

Usage

In the painter Select the control and set the value in the Properties view, Position tab, Autosize Height check box.

Minimum height The height of the column, computed field, or text will never be less than the minimum height (the height selected in the painter).

Examples

```
string setting
setting = dw_1.Object.empname.Height.AutoSize

dw_1.Object.empname.Height.AutoSize = "Yes"

setting = dw_1.Describe("empname.Height.AutoSize")

dw_1.Modify("empname.Height.AutoSize=Yes")
```

Help.*property*

Description

Settings for customizing the Help topics associated with DataWindow dialog boxes.

PocketBuilder	✗
PowerBuilder	✓

For more information about Help, see the ShowHelp function in the *PowerScript Reference*.

Applies to

DataWindows

Syntax

PowerBuilder dot notation:

dw_control.Object.DataWindow.Help.*property*

Describe and Modify argument:

"DataWindow.Help.*property* { = *value* }"

HideGrayLine

Description

Shows or hides a gray line to indicate that a fixed page has been crossed when scrolling in a DataWindow with group headers.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

DataWindow control

Syntax

PocketBuilder dot notation:

dw_control.Object.DataWindow.HideGrayLine

Describe and Modify argument:

"DataWindow.HideGrayLine { = ' *value* ' }"

Parameter	Description
value	(*exp*) Whether a gray line displays in the Preview view and at runtime. Values are: Yes — The gray line is hidden. No — The gray line displays (default). *Value* can be a quoted DataWindow expression.

Usage

This property can be set in the open event for the window in which the DataWindow displays. Note that you cannot suppress the display of repeating group headers.

In the painter Select the DataWindow object by deselecting all controls; then set the value in the Properties view, General tab. This option is enabled only for DataWindows with group headers.

Examples

```
dw_1.Object.DataWindow.HideGrayLine = yes
```

HideSnaked

Description

Whether the control appears only once per page when you print the DataWindow using the newspaper columns format.

PocketBuilder	✗
PowerBuilder	✓

Applies to

Button, Column, Computed Field, Graph, GroupBox, Line, OLE, Oval, Picture, Rectangle, Report, RoundRectangle, TableBlob, and Text controls

Syntax

PowerBuilder dot notation:

dw_control.Object.*controlname*.HideSnaked

Describe and Modify argument:

"*controlname*.HideSnaked { = ' *value* ' }"

Horizontal_Spread

Description

The space between columns in the detail area of the DataWindow object. Horizontal_Spread is meaningful *only* when type is Grid or Tabular.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Style keywords

Syntax

SyntaxFromSQL:

Style (Horizontal_Spread = *value*)

Parameter	Description
value	An integer specifying the space between columns in the detail area of the DataWindow object area in the units specified for the DataWindow

Examples

```
SQLCA.SyntaxFromSQL(sqlstring, &
'Style(...Horizontal_Spread = 25 ...)', errstring)
```

HorizontalScrollMaximum

Description

The maximum width of the scroll box of the DataWindow's horizontal scroll bar. This value is set by PocketBuilder based on the layout of the DataWindow object and the size of the DataWindow control. Use HorizontalScrollMaximum with HorizontalScrollPosition to synchronize horizontal scrolling in multiple DataWindow objects.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

DataWindows

Syntax

PocketBuilder dot notation:

dw_control.Object.DataWindow.HorizontalScrollMaximum

Describe argument:

"DataWindow.HorizontalScrollMaximum"

Examples

```
string setting
setting = &
dw_1.Object.DataWindow.HorizontalScrollMaximum

setting = &
dw_1.Describe("DataWindow.HorizontalScrollMaximum")
```

HorizontalScrollMaximum2

Description

The maximum width of the second scroll box when the horizontal scroll bar is split (HorizontalScrollSplit is greater than 0). This value is set by PocketBuilder based on the content of the DataWindow. Use HorizontalScrollMaximum2 with HorizontalScrollPosition2 to synchronize horizontal scrolling in multiple DataWindow objects.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

DataWindows

Syntax

PocketBuilder dot notation:

dw_control.Object.DataWindow.HorizontalScrollMaximum2

Describe argument:

"DataWindow.HorizontalScrollMaximum2"

Examples

```
string setting
setting = &
dw_1.Object.DataWindow.HorizontalScrollMaximum2
```

```
setting = &
dw_1.Describe("DataWindow.HorizontalScrollMaximum2")
```

HorizontalScrollPosition

Description

The position of the scroll box in the horizontal scroll bar. Use HorizontalScrollMaximum with HorizontalScrollPosition to synchronize horizontal scrolling in multiple DataWindow objects.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

DataWindows

Syntax

PocketBuilder dot notation:

dw_control.Object.DataWindow.HorizontalScrollPosition

Describe and Modify argument:

"DataWindow.HorizontalScrollPosition { = *scrollvalue* }"

Parameter	Description
scrollvalue	An integer specifying the position of the scroll box in the horizontal scroll bar of the DataWindow

Examples

```
string spos1
spos1 =
dw_1.Object.DataWindow.HorizontalScrollPosition
```

```
string smax1, smax2, spos1, modstring
integer pos2
smax1 = dw_1.Describe( &
"DataWindow.HorizontalScrollMaximum")
spos1 = dw_1.Describe( &
```

```
"DataWindow.HorizontalScrollPosition")
smax2 = dw_2.Describe( &
"DataWindow.HorizontalScrollMaximum")
pos2 = Integer(spos1) * Integer(smax2) / Integer(smax1)
modstring = "DataWindow.HorizontalScrollPosition=" &
+ String(pos2)
dw_1.Modify(modstring)
```

HorizontalScrollPosition2

Description

The position of the scroll box in the second portion of the horizontal scroll bar when the scroll bar is split (HorizontalScrollSplit is greater than 0). Use HorizontalScrollMaximum2 with HorizontalScrollPosition2 to synchronize horizontal scrolling in multiple DataWindow objects.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

DataWindows

Syntax

PocketBuilder dot notation:

dw_control.Object.DataWindow.HorizontalScrollPosition2

Describe and Modify argument:

"DataWindow.HorizontalScrollPosition2 { = *scrollvalue* }"

Parameter	Description
scrollvalue	An integer specifying the position of the scroll box in the second portion of a split horizontal scroll bar of the DataWindow

Examples

```
string spos
spos =
dw_1.Object.DataWindow.HorizontalScrollPosition2

dw_1.Object.DataWindow.HorizontalScrollPosition2 = 200

spos = dw_1.Describe( &
"DataWindow.HorizontalScrollPosition2")

dw_1.Modify("DataWindow.HorizontalScrollPosition2=200"
)
```

HorizontalScrollSplit

Description

The position of the split in the DataWindow's horizontal scroll bar. If HorizontalScrollSplit is zero, the scroll bar is not split.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

DataWindows

Syntax

PocketBuilder dot notation:

dw_control.Object.DataWindow.HorizontalScrollSplit

Describe and Modify argument:

"DataWindow.HorizontalScrollSplit { = *splitdistance* }"

Parameter	Description
splitdistance	An integer indicating where the split will occur in the horizontal scroll bar in a DataWindow object in the unit of measure specified for the DataWindow object

Examples

```
string setting
setting = dw_1.Object.DataWindow.HorizontalScrollSplit

dw_1.Object.DataWindow.HorizontalScrollSplit = 250

setting = &
dw_1.Describe("DataWindow.HorizontalScrollSplit")

dw_1.Modify("DataWindow.HorizontalScrollSplit=250")
```

HTextAlign

Description

The way text in a button is horizontally aligned.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Button controls

Syntax

PocketBuilder dot notation:

dw_control.Object.*buttonname*.HTextAlign

Describe and Modify argument:

"*buttonname*.HTextAlign { = ' *value* ' }"

Parameter	Description
buttonname	The name of the button for which you want to align text.
value	An integer indicating how the button text is horizontally aligned. Values are: 0 — Center 1 — Left 2 — Right

Usage

In the painter Select the control and set the value in the Properties view, General tab, Horizontal Alignment option.

Examples

```
dw_1.Object.b_name.HTextAlign = "1"

setting = dw_1.Describe("b_name.HTextAlign")

dw_1.Modify("b_name.HTextAlign ='1'")
```

HTML.*property*

Description

Settings for adding user-defined HTML syntax and hyperlinks to controls in a Web DataWindow.

PocketBuilder	✗
PowerBuilder	✓

Applies to

Column, Computed Field, Picture, and Text controls

Syntax

PowerBuilder dot notation:

dw_control.Object.*controlname*.HTML.*property*

Describe and Modify argument:

"*controlname*.HTML.*property* { = ' *value* ' }"

HTMLDW

Description

Specifies whether HTML generated for the DataWindow object provides updates and interactivity.

PocketBuilder	✗
PowerBuilder	✓

Applies to

DataWindow objects

Syntax

PowerBuilder dot notation:

dw_control.Object.DataWindow.HTMLDW = *value*

Describe and Modify argument:

"DataWindow.HTMLDW { = ' *value* ' }"

HTMLGen.*property*

Description

Settings that control the level of features incorporated into HTML generated for the DataWindow.

PocketBuilder	✗
PowerBuilder	✓

Applies to

DataWindow objects

Syntax

PowerBuilder dot notation:

dw_control.Object.DataWindow.HTMLGen.*property*

Describe and Modify argument:

"DataWindow.HTMLGen.*property* { = ' *value* ' }"

HTMLTable.*property*

Description

Settings for the display of DataWindow data when displayed in HTML table format. These settings simplify the transfer of data from a database to an HTML page. They are particularly useful when used to create HTML pages dynamically.

PocketBuilder	✗
PowerBuilder	✓

Applies to
DataWindow objects

Syntax
PowerBuilder dot notation:

dw_control.Object.DataWindow.HTMLTable.*property*

Describe and Modify argument:

"DataWindow.HTMLTable.*property* { = ' *value* ' }"

ID

Description
The number of the column or TableBlob.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to
Column and TableBlob controls

Syntax
PocketBuilder dot notation:

dw_control.Object.*controlname*.ID

Describe and Modify argument:

"*controlname*.ID"

Parameter	Description
controlname	The name of the column or TableBlob for which you want the ID number

Examples

```
setting = dw_1.Object.empname.ID
setting = dw_1.Describe("empname.ID")
```

Identity

Description

Whether the database is to supply the value of the column in a newly inserted row. If so, the column is not updatable; the column is excluded from the INSERT statement.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Not all DBMSs support the identity property. For more information see the documentation for your DBMS.

Applies to

Column controls

Syntax

PocketBuilder dot notation:

dw_control.Object.*columnname*.Identity

Describe and Modify argument:

"*columnname*.Identity { = ' *value* ' }"

Parameter	Description
columnname	A string containing the name of the column for which you want to get or set the identity property.
value	A string indicating whether a column's value in a newly inserted row is supplied by the DBMS: Yes — The DBMS will supply the value of the column in a newly inserted row; the column is not updatable. No — The column is updatable.

Examples

```
dw_1.Object.empid.Identity = "yes"

dw_1.Modify("empid.Identity='yes'")
```

Import.XML.Trace

Description

Setting that determines whether import trace information is written to a log file.

PocketBuilder	✗
PowerBuilder	✓

Applies to

DataWindow objects

Syntax

PowerBuilder dot notation:

dw_control.Object.DataWindow.Import.XML.Trace

Describe and Modify argument:

"DataWindow.Import.XML.Trace { = ' *value* ' }"

Import.XML.TraceFile

Description

Specifies the name and location of an import trace file.

PocketBuilder	✗
PowerBuilder	✓

Applies to

DataWindow objects

Syntax

PowerBuilder dot notation:

dw_control.Object.DataWindow.Import.XML.TraceFile

Describe and Modify argument:

"DataWindow.Import.XML.TraceFile { = ' *value* ' }"

Import.XML.UseTemplate

Description

Setting that optionally controls the logical structure of the XML imported from an XML file into a DataWindow object using the ImportFile method.

PocketBuilder	✗
PowerBuilder	✓

Applies to

DataWindow objects

Syntax

PowerBuilder dot notation:

dw_control.Object.DataWindow.Import.XML.UseTemplate

Describe and Modify argument:

"DataWindow.Import.XML.UseTemplate { = ' *value* ' }"

Initial

Description

The initial value of the column in a newly inserted row.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Column controls

Syntax

PocketBuilder dot notation:

dw_control.Object.*columnname*.Initial

Describe and Modify argument:

"*columnname*.Initial { = ' *initialvalue* ' }"

Parameter	Description
columnname	A string containing the name of the column for which you want to get or set the initial property.
initialvalue	A string containing the initial value of the column. Special values include: Empty — A string of length 0 Null — No value Spaces — All blanks Today — Current date, time, or date and time

Examples

```
setting = dw_1.Object.empname.Initial

dw_1.Object.empname.Initial = "empty"

setting = dw_1.Describe("empname.Initial")

dw_1.Modify("empname.Initial='empty'")

dw_1.Modify("empstatus.Initial='A'")
```

Invert

Description

The way the colors in a Picture control are displayed, either inverted or normal.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Picture controls

Syntax

PocketBuilder dot notation:

dw_control.Object.*bitmapname*.Invert

Describe and Modify argument:

"*bitmapname*.Invert { = ' *number* ' }"

Parameter	Description
bitmapname	The name of the Picture control in the DataWindow for which you want to invert the colors.
number	(*exp*) A boolean number indicating whether the colors of the picture will display inverted. Values are: 0 — (Default) No; do not invert the picture's colors. 1 — Yes; display the picture with colors inverted. *Number* can be a quoted DataWindow expression.

Usage

In the painter Select the control and set the value in the Properties view, General tab, Invert Image check box.

Examples

```
string setting
setting = dw_1.Object.bitmap_1.Invert

dw_1.Object.bitmap_1.Invert="0~tIf(empstatus='A',0,1)"

setting = dw_1.Describe("bitmap_1.Invert")

dw_1.Modify( &
"bitmap_1.Invert='0~tIf(empstatus=~~~'A~~~',0,1)'")
```

Key

Description

Whether the column is part of the database table's primary key.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Column controls

Syntax

PocketBuilder dot notation:

dw_control.Object.*columnname*.Key

Describe and Modify argument:

"*columnname*.Key { = *value* }"

Parameter	Description
columnname	The column for which you want to get or set primary key status.
value	Whether the column is part of the primary key. Values are: Yes — The column is part of the primary key. No — The column is not part of the key.

Usage

In the painter Set the value using the Rows menu, Update Properties.

Examples

```
string setting
setting = dw_1.Object.empid.Key

dw_1.Object.empid.Key = "Yes"

setting = dw_1.Describe("empid.Key")

dw_1.Modify("empid.Key=Yes")
```

KeyClause

Description

An expression to be used as the key clause when retrieving the blob.

PocketBuilder	✗
PowerBuilder	✓

Applies to

TableBlob controls

Syntax

PowerBuilder dot notation:

dw_control.Object.*tblobname*.KeyClause

Describe and Modify argument:

"*tblobname*.KeyClause { = ' *keyclause* ' }"

Label.*property*

Description

Settings for a DataWindow whose presentation style is Label.

PocketBuilder	✗
PowerBuilder	✓

Applies to

DataWindows

Syntax

PowerBuilder dot notation:

dw_control.Object.DataWindow.Label.*property*

Describe and Modify argument:

"DataWindow.Label.*property* { = *value* }"

SyntaxFromSQL:

DataWindow(Label.*property* = *value*)

LabelDispAttr.*fontproperty*

See DispAttr.fontproperty.

LastRowOnPage

Description

The last row currently visible in the DataWindow.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

DataWindows

Syntax

PocketBuilder dot notation:

dw_control.Object.DataWindow.LastRowOnPage

Describe argument:

"DataWindow.LastRowOnPage"

Examples

```
string setting
setting = dw_1.Object.DataWindow.LastRowOnPage

setting = dw_1.Describe("DataWindow.LastRowOnPage")
```

Left_Margin

Description

The size of the left margin of the DataWindow object.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Style keywords

Syntax

SyntaxFromSQL:

Style (Left_Margin = *value*)

Parameter	Description
value	An integer specifying the size of the left margin in the units specified for the DataWindow.

Examples

```
SQLCA.SyntaxFromSQL(sqlstring, &
'Style( ... LeftMargin = 500 ... )', errstring)
```

Legend

Description

The location of the legend in a Graph control in a DataWindow.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Graph controls

Syntax

PocketBuilder dot notation:

dw_control.Object.*graphname*.Legend

Describe and Modify argument:

"*graphname*.Legend { = ' *value* ' }"

Parameter	Description
graphname	The name of the graph control for which you want to specify the location of the legend.

Parameter	Description
value	(*exp*) A number indicating the location of the legend of a graph. Values are: 0 — None 1 — Left 2 — Right 3 — Top 4 — Bottom *Value* can be a quoted DataWindow expression.

Usage

In the painter Select the control and set the value in the Properties view, General tab, Legend option (applicable when the graph has more than one series).

Examples

```
string setting
setting = dw_1.Object.graph_1.Legend

dw_1.Object.graph_1.Legend = 2

setting = dw_1.Describe("graph_1.Legend")

dw_1.Modify("graph_1.Legend=2")

dw_1.Modify("graph_1.Legend='2~tIf(dept_id=200,0,2)'")
```

Legend.DispAttr.*fontproperty*

See DispAttr.fontproperty.

Level

Description

The grouping level.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Level is only used in DataWindow syntax for the Create function.

Applies to

Group keywords

Syntax

Group (BY(*colnum1*, *colnum2*, ...) ... Level = *n* ...)

LineRemove

Description

(RichText presentation style only) Whether the line of text that contains the input field for the column or computed field is removed when the input field is empty. LineRemove is similar to the SlideUp property for controls in other presentation styles.

PocketBuilder	✗
PowerBuilder	✓

Applies to

Column and Computed Field controls in the RichText presentation style

Syntax

PowerBuilder dot notation:

dw_control.Object.*controlname*.LineRemove

Describe and Modify argument:

"*controlname*.LineRemove { = ' *value* ' }"

LinkUpdateOptions

Description

When the OLE Object control is linked, the method for updating the link information. If the user tries to activate the OLE object and PowerBuilder cannot find the linked file, which breaks the link, LinkUpdateOptions controls whether PowerBuilder automatically displays a dialog box prompting the user to find the file. If you turn off the automatic dialog box, you can reestablish the link by calling the LinkTo or LinkUpdateDialog in a script.

PocketBuilder	✗
PowerBuilder	✓

Applies to

OLE Object controls

Syntax

PowerBuilder dot notation:

dw_control.Object.*olecontrolname*.LinkUpdateOptions

Describe and Modify argument:

"*olecontrolname*.LinkUpdateOptions { = ' *updatetype* ' }"

Message.Title

Description

The title of the dialog box that displays when an error occurs.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

DataWindows

Syntax

PocketBuilder dot notation:

dw_control.Object.DataWindow.Message.Title

Describe and Modify argument:

"DataWindow.Message.Title { = ' *titlestring* ' }"

SyntaxFromSQL:

DataWindow(Message.Title = ' *titlestring* ')

Parameter	Description
titlestring	A string containing the title for the title bar of the DataWindow dialog box that displays when an error occurs.

Examples

```
setting = dw_1.Object.DataWindow.Message.Title

dw_1.Object.DataWindow.Message.Title = "Mistake!"

setting = dw_1.Describe("DataWindow.Message.Title")

dw_1.Modify("DataWindow.Message.Title='Bad, Bad,
Bad'")

SQLCA.SyntaxFromSQL(sql_syntax, &
"Style(...) &
DataWindow(Message.Title='Sales Report' ...) ...", &
ls_Errors)
```

Moveable

Description

Whether the specified control in the DataWindow can be moved during execution. Moveable controls should be in the DataWindow's foreground.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Button, Column, Computed Field, Graph, GroupBox, Line, OLE, Oval, Picture, Rectangle, Report, RoundRectangle, TableBlob, and Text controls

Syntax

PocketBuilder dot notation:

dw_control.Object.*controlname*.Moveable

Describe and Modify argument:

"*controlname*.Moveable { = *number* }"

Parameter	Description
controlname	The control within the DataWindow for which you want to get or set the Moveable property that governs whether the user can move the control.
number	A boolean number specifying whether the control is movable. Values are: 0 — False, the control is not movable. 1 — True, the control is movable.

Usage

In the painter Select the control and set the value in the Properties view, Position tab.

Examples

```
string setting
setting = dw_1.Object.bitmap_1.Moveable

dw_1.Object.bitmap_1.Moveable = 1

setting = dw_1.Describe("bitmap_1.Moveable")

dw_1.Modify("bitmap_1.Moveable=1")
```

Multiline

Description

(RichText presentation style) Whether the column or computed field can contain multiple lines. Multiline is only effective when Width.Autosize is set to No.

PocketBuilder	✗
PowerBuilder	✓

Applies to

Column and Computed Field controls in the RichText presentation style

Syntax

PowerBuilder dot notation:

dw_control.Object.*controlname*.Multiline

Describe and Modify argument:

"*controlname*.Multiline { = ' *value* ' }"

Name

Description

The name of the control.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Button, Column, Computed Field, Graph, GroupBox, Line, OLE, Oval, Picture, Rectangle, Report, RoundRectangle, TableBlob, and Text controls

Syntax

PocketBuilder dot notation:

dw_control.Object.*controlname*.Name

Describe argument:

"*controlname*.Name"

Parameter	Description
controlname	The control for which you want the name. For columns, you can specify the column number preceded by #.

Usage

In the painter Select the control and set the value in the Properties view, General tab, Name option.

Examples

```
setting = dw_1.Object.#4.Name
setting = dw_1.Describe("#4.Name")
```

Nest_Arguments

Description

The values for the retrieval arguments of a nested report. The number of values in the list should match the number of retrieval arguments defined for the nested report.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Report controls

Syntax

PocketBuilder dot notation:

dw_control.Object.*reportname*.Nest_Arguments

Describe and Modify argument:

"*reportname*.Nest_Arguments { = *list* } "

Parameter	Description
reportname	The name of the nested report for which you want to supply retrieval argument values.
list	A list of values for the retrieval arguments of the nested report. The format for the list is: (("*arg1*") {,("*arg2*") {,("*arg3*") {,... } } }) The list is not a quoted string. It is surrounded by parentheses, and each argument value within the list is parenthesized, surrounded with double quotes, and separated by commas. If an argument is a literal string, use single quotes within the double quotes. When changing the values for the retrieval arguments, you must supply values for all the retrieval arguments defined for the report. If you specify fewer or more arguments, an error will occur during execution when the DataWindow retrieves its data. To remove the report's retrieval arguments, specify empty parentheses. If no arguments are specified, the user is prompted for the values during execution.

Usage

In the painter Select the control and set the value in the Properties view, Arguments tab.

Examples

```
setting = dw_1.Object.rpt_1.Nest_Arguments

dw_1.Object.rpt_1.Nest_Arguments = &
"((~"cust_id~"),(~"'Eastern'~"))"
```

```
setting = dw_1.Describe("rpt_1.Nest_Arguments")

dw_1.Modify("rpt_1.Nest_Arguments" &
"=((~"cust_id~"),(~"'Eastern'~"))")

dw_1.Modify("rpt_1.Nest_Arguments=()")
```

Nested

Description

Whether the DataWindow contains nested DataWindows. Values returned are Yes or No.

PocketBuilder	✗
PowerBuilder	✓

Applies to

DataWindows

Syntax

PowerBuilder dot notation:

dw_control.Object.DataWindow.Nested

Describe argument:

"DataWindow.Nested"

NewPage (Group keywords)

Description

Whether a change in the value of a group column causes a page break.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Group keywords

Syntax

SyntaxFromSQL:

Group (*colnum1*, *colnum2* NewPage)

Examples

```
SQLCA.SyntaxFromSQL(sql_syntax, &
"Style(Type=Group) " + &
"Group(#3 NewPage ResetPageCount)", &
ls_Errors)
```

NewPage (Report controls)

Description

Whether a nested report starts on a new page. NewPage applies only to reports in a composite DataWindow. Note that if the Trail_Footer property of the preceding report is set to No, the current report will be forced to begin on a new page regardless of the NewPage value.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Report controls

Syntax

PocketBuilder dot notation:

dw_control.Object.*reportname*.NewPage

Describe and Modify argument:

"*reportname*.NewPage { = *value* } "

Parameter	Description
reportname	The name of the report control for which you want to get or set the NewPage property.
value	Whether the report begins a new page. Values are: Yes — Start the report on a new page. No — Do not start the report on a new page.

Usage

In the painter Select the Report control in the Composite presentation style and set the value in the Properties view, General tab, New Page check box.

Examples

```
string newpage_setting
newpage_setting = dw_1.Object.rpt_1.NewPage

dw_1.Object.rpt_1.NewPage = "Yes"

newpage_setting = dw_1.Describe("rpt_1.NewPage")

dw_1.Modify("rpt_1.NewPage=Yes")
```

NoUserPrompt

Description

Determines whether message boxes are displayed to the user during DataWindow processing.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

DataWindows

Syntax

PocketBuilder dot notation:

dw_control.Object.DataWindow.NoUserPrompt

Describe and Modify argument:

"DataWindow.NoUserPrompt { = ' *value* ' }"

Parameter	Description
value	A string specifying whether any message box requiring user intervention displays during DataWindow processing. Values are: Yes — No message box displays. No — (Default) Message boxes display when invoked during DataWindow processing.

Usage

Set the NoUserPrompt property to yes if the DataWindow is to be used in a batch process or in an EAServer environment when there is no possibility of end user intervention. Dialog boxes you can prevent from displaying include the Error, Print, Retrieve, CrossTab, Expression, SaveAs, Import, Query, RichText, Filter, and Sort dialog boxes.

Examples

```
dw_1.Object.DataWindow.NoUserPrompt = "yes"
dw_1.Modify("DataWindow.NoUserPrompt=no")
```

Objects

Description

A list of the controls in the DataWindow object. The names are returned as a tab-separated list.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

DataWindows

Syntax

PocketBuilder dot notation:

dw_control.Object.DataWindow.Objects

Describe argument:

"DataWindow.Objects"

Examples

```
setting = dw_1.Describe("DataWindow.Objects")
```

OLE.Client.*property*

Description

Settings that some OLE server applications use to identify the client's information. The property values can be used to construct the title of the server window.

PocketBuilder	✗
PowerBuilder	✓

Applies to

DataWindows

Syntax

PowerBuilder dot notation:

dw_control.Object.DataWindow.OLE.Client.*property*

Describe and Modify argument:

"DataWindow.OLE.Client.*property* { = ' *value* ' }"

OLEClass

Description

The name of the OLE class for the TableBlob control.

PocketBuilder	✗
PowerBuilder	✓

Applies to

TableBlob controls

Syntax

PowerBuilder dot notation:

dw_control.Object.*tblobname*.OLEClass

Describe and Modify argument:

"*tblobname*.OLEClass { = ' *oleclassname* ' }"

OverlapPercent

Description

The percentage of overlap for the data markers (such as bars or columns) in different series in a graph.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Graph controls

Syntax

PocketBuilder dot notation:

dw_control.Object.*graphname*.OverlapPercent

Describe and Modify argument:

"*graphname*.OverlapPercent { = ' *integer* ' }"

Parameter	Description
graphname	The name of the graph control in the DataWindow object for which you want to get or set the percentage of overlap.
integer	(*exp*) An integer specifying the percent of the width of the data markers that will overlap. *Integer* can be a quoted DataWindow expression.

Usage

In the painter Select the control and set the value in the Properties view, Graph tab, OverlapPercent option (applicable when a series has been specified).

Examples

```
string setting
setting = dw_1.Object.graph_1.OverlapPercent

dw_1.Object.graph_1.OverlapPercent = 25

setting = dw_1.Describe("graph_1.OverlapPercent")

dw_1.Modify("graph_1.OverlapPercent=25")
```

Pen.*property*

Description

Settings for a line or the outline of a control.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Line, Oval, Rectangle, and RoundRectangle controls

Syntax

PocketBuilder dot notation:

dw_control.Object.*controlname*.Pen.*property*

Describe and Modify argument:

"*controlname*.Pen.*property* { = *value* }"

Parameter	Description
controlname	The name of the control whose Pen property you want to get or set.
property	A property that applies to the Pen characteristics of *controlname*, as listed in the table below.
value	The value of the property, as shown in the table below. *Value* can be a quoted DataWindow expression.

Property for Pen	Value
Color	(*exp*) A long specifying the color (the red, green, and blue values) to be used as the control's line color. Painter: Pen Color option.
Style	(*exp*) A number specifying the style of the line. Values are: 0 — Solid 1 — Dash 2 — Dotted 3 — Dash-dot pattern 4 — Dash-dot-dot pattern 5 — Null (no visible line) Painter: Pen Style option.
Width	(*exp*) A number specifying the width of the line in the unit of measure specified for the DataWindow. Painter: Pen Width option (not available when Style is a value other than Solid).

Usage

In the painter Select the control and set values in the Properties view, General tab.

Examples

```
string setting
setting = dw_1.Object.line_1.Pen.Width
dw_1.Object.line_1.Pen.Width = 10
setting = dw_1.Describe("line_1.Pen.Width")
dw_1.Modify("line_1.Pen.Width=10")
```

Perspective

Description

The distance the graph appears from the front of the window.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Graph controls

Syntax

PocketBuilder dot notation:

dw_control.Object.*graphname*.Perspective

Describe and Modify argument:

"*graphname*.Perspective { = ' *integer* ' }"

Parameter	Description
graphname	The name of the graph control in the DataWindow object for which you want to get or set the perspective.
integer	(*exp*) An integer between 1 and 100 specifying how far away the graph appears. The larger the number, the greater the distance and the smaller the graph appears. *Integer* can be a quoted DataWindow expression.

Usage

In the painter Select the control and set the value in the Properties view, Graph tab, Perspective scroll bar (available when a 3D graph type is selected).

Examples

```
string setting
setting = dw_1.Object.graph_1.Perspective

dw_1.Object.graph_1.Perspective = 20

setting = dw_1.Describe("graph_1.Perspective")

dw_1.Modify("graph_1.Perspective=20")
```

Pie.DispAttr.*fontproperty*

See DispAttr.fontproperty.

Pointer

Description

The image to be used for the mouse pointer when the pointer is over the specified control. If you specify a pointer for the whole DataWindow, PowerBuilder uses that pointer except when the pointer is over a control that also has a Pointer setting.

PocketBuilder	✗
PowerBuilder	✓

Applies to

DataWindow, Button, Column, Computed Field, Graph, GroupBox, Line, Oval, Picture, Rectangle, Report, RoundRectangle, TableBlob, and Text controls

Syntax

PowerBuilder dot notation:

dw_control.Object.*controlname*.Pointer

Describe and Modify argument:

"*controlname*.Pointer { = ' *pointername* ' }"

Print.Buttons

Description

Whether buttons display on the printed output.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

DataWindows

Syntax

PocketBuilder dot notation:

dw_control.Object.DataWindow.Print.Buttons

Describe and Modify argument:

"DataWindow.Print.Buttons { = *value* }"

Parameter	Description
value	Whether buttons display on the printed output. Values are: Yes — Buttons are displayed. No — Buttons are not displayed.

Usage

In the painter Select the DataWindow by deselecting all controls; then set the value in the Properties view, Print Specifications tab.

Examples

```
dw_1.Object.DataWindow.Print.Buttons = 'Yes'

setting = dw_1.Describe("DataWindow.Print.Buttons")

dw_1.Modify("DataWindow.Print.Buttons = 'Yes'")
```

Print.Preview.Buttons

Description

Whether buttons display in print preview.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

DataWindows

Syntax

PocketBuilder dot notation:

dw_control.Object.DataWindow.Print.Preview.Buttons

Describe and Modify argument:

"DataWindow.Print.Preview.Buttons { = *value* }"

Parameter	Description
value	Whether buttons display in print preview. Values are: Yes — Buttons are displayed. No — Buttons are not displayed.

Usage

In the painter Select the DataWindow by deselecting all controls; then set the value in the Properties view, Print Specification tab.

Examples

```
dw_1.Object.DataWindow.Print.Preview.Buttons = 'Yes'

setting = dw_1.Describe
       ("DataWindow.Print.Preview.Buttons")

dw_1.Modify("DataWindow.Print.Preview.Buttons =
'Yes'")
```

Print.*property*

Description

Properties that control the printing of a DataWindow.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

DataWindows

Syntax

PocketBuilder dot notation:

dw_control.Object.DataWindow.Print.*property*

Describe and Modify argument:

"DataWindow.Print.*property* { = *value* }"

SyntaxFromSQL:

DataWindow (Print.*property* = *value*)

Parameter	Description
property	A property for printing. Properties and their settings are listed in the table below.
value	The value to be assigned to the property. *Value* cannot be a DataWindow expression.

Property for Print	Value
CanUseDefault Printer	Whether a report can be printed on the default system printer if the printer specified by the PrinterName property is not valid. Painter: Can Use Default Printer option.
ClipText	Whether the text of a static text field on a printed page is clipped to the dimensions of the text field when the text field has no visible border setting. Values are Yes — The printed text does not overrun the text field. No — (Default) The entire text can overrun the text field. Text is automatically clipped for text fields with visible border settings even if this property is not set. Painter: Clip Text option.
Collate	Whether printing is collated. Note that collating is usually slower since the print is repeated to produce collated sets. Values are: Yes — (Default) Collate the pages of the print job. No — Do not collate. Painter: Collate Copies option.

Property for Print	Value
Color	An integer indicating whether the printed output will be color or monochrome. Values are: 1 — Color 2 — Monochrome The user can specify the value in the system's Print dialog box if the printer driver supports it.
Columns	An integer specifying the number of newspaper-style columns the DataWindow will print on a page. For purposes of page fitting, the whole DataWindow is a single column. The default is 1. Painter: Newspaper Columns Across option.
Columns.Width	An integer specifying the width of the newspaper-style columns in the units specified for the DataWindow. Painter: Newspaper Columns Width option.
Copies	An integer indicating the number of copies to be printed. The user can also specify this value in the system's Print Setup dialog box if the printer driver supports it. If you use *both* the Print.Copies property and the Print Setup dialog box to indicate that multiple copies should be printed, the total number of copies printed is the product of both values.
CustomPage.Length	A double indicating the desired length of a custom paper size for printing. Use this property in conjunction with Print.CustomPage.Width and with Paper.Size set to 256.
CustomPage.Width	A double indicating the desired width of a custom paper size for printing. Use this property in conjunction with Print.CustomPage.Length and with Paper.Size set to 256.
DocumentName	A string containing the name that will display in the print queue when the user sends the contents of the DataWindow object to the printer. Painter: Document Name option.
Duplex	An integer indicating the orientation of the printed output. Values are: 1 — Simplex (none) 2 — Horizontal 3 — Vertical The user can specify the value in the system's Print dialog box if the printer driver supports it.

Property for Print	Value
Filename	A string containing the name of the file to which you want to print the report. An empty string means send to the printer. Painter: Not settable in painter.
Margin.Bottom	An integer indicating the width of the bottom margin on the printed page in the units specified for the DataWindow. You can set Margin.Bottom when using SyntaxFromSQL to generate DataWindow syntax. Painter: Bottom Margin option.
Margin.Left	An integer indicating the width of the left margin on the printed page in the units specified for the DataWindow. You can set Margin.Left when using SyntaxFromSQL to generate DataWindow syntax. Painter: Left Margin option.
Margin.Right	An integer indicating the width of the right margin on the printed page in the units specified for the DataWindow. You can set Margin.Right when using SyntaxFromSQL to generate DataWindow syntax. Painter: Right Margin option.
Margin.Top	An integer indicating the width of the top margin on the printed page in the units specified for the DataWindow. You can set Margin.Top when using SyntaxFromSQL to generate DataWindow syntax. Painter: Top Margin option.
Orientation	An integer indicating the print orientation. Values are: 0 — The default orientation for your printer 1 — Landscape 2 — Portrait Painter: Paper Orientation option.

Property for Print	Value
OverridePrintJob	Whether you want to override the print job print settings defined in the PrintOpen method with the print specifications of the DataWindow. Values are: Yes — Override the print job print settings. No — (Default) Do not override the print job print settings. Painter: Override Print Job option.
Page.Range	A string containing the numbers of the pages you want to print, separated by commas. You can also specify a range with a dash. For example, to print pages 1, 2, and 5 through 10, enter: "1,2, 5-10". The empty string means print all. The user can specify the value in the system's Print dialog box if the printer driver supports it.
Page.RangeInclude	An integer indicating what pages to print within the desired range. Values are: 0 — Print all. 1 — Print all even pages. 2 — Print all odd pages. The user can specify the value in the system's Print dialog box if the printer driver supports it.

Property for Print	Value
Paper.Size	An integer indicating the size of the paper used for the output: 0 — Default paper size for the printer 1 — Letter 8 1/2 x 11 in 2 — LetterSmall 8 1/2 x 11in 3 — Tabloid 17 x 11 inches 4 — Ledger 17 x 11 in 5 — Legal 8 1/2 x 14 in 6 — Statement 5 1/2 x 8 1/2 in 7 — Executive 7 1/4 x 10 1/2 in 8 — A3 297 x 420 mm 9 — A4 210 x 297 mm 10 — A4 Small 210 x 297 mm 11 — A5 148 x 210 mm 12 — B4 250 x 354 mm 13 — B5 182 x 257 mm 14 — Folio 8 1/2 x 13 in 15 — Quarto 215 x 275mm 16 — 10x14 in 17 — 11x17 in 18 — Note 8 1/2 x 11 in 19 — Envelope #9 3 7/8 x 8 7/8 20 — Envelope #10 4 1/8 x 9 1/2 21 — Envelope #11 4 1/2 x 10 3/8 22 — Envelope #12 4 x 11 1/276 23 — Envelope #14 5 x 11 1/2 24 — C size sheet 25 — D size sheet 26 — E size sheet 27 — Envelope DL 110 x 220mm 28 — Envelope C5 162 x 229 mm 29 — Envelope C3 324 x 458 mm 30 — Envelope C4 229 x 324 mm 31 — Envelope C6 114 x 162 mm 32 — Envelope C65 114 x 229 mm 33 — Envelope B4 250 x 353 mm 34 — Envelope B5 176 x 250 mm 35 — Envelope B6 176 x 125 mm 36 — Envelope 110 x 230 mm 37 — Envelope Monarch 3.875 x 7.5 in 38 — 6 3/4 Envelope 3 5/8 x 6 1/2 in 39 — US Std Fanfold 14 7/8 x 11 in 40 — German Std Fanfold 8 1/2 x 12 in 41 — German Legal Fanfold 8 1/2 x 13 in 256 — User-defined paper size

Property for Print	Value

	To specify a user-defined paper size, set the Paper.Size property to 256, then set the Print.CustomPage.Length and Print.Custom.Page.Width properties to the desired size in millimeters. For example:
	`dw_1.Object.DataWindow.Print.Paper.Size = 256`
	`dw_1.Object.DataWindow.Print.CustomPage.Length = 254 //10 inches`
	`dw_1.Object.DataWindow.Print.CustomPage.Width = 190.5 //7 inches`
	Painter: Paper Size option.

Paper.Source	An integer indicating the bin that will be used as the paper source. The integer you use depends on the tray number used by the printer. (To determine the actual bin setting, you can query the printer with a utility that makes API calls to the printer driver.) Typical values are: 0 — Default 1 — Upper 2 — Lower 3 — Middle 4 — Manual 5 — Envelope 6 — Envelope manual 7 — Auto 8 — Tractor 9 — Smallfmt 10 — Largefmt 11 — Large capacity 14 — Cassette Painter: Paper Source option.
Preview	Whether the DataWindow object is displayed in preview mode. Values are: Yes — Display in preview mode. No — (Default) Do not display in preview mode.
Preview.Rulers	Whether the rulers display when the DataWindow object displays in preview mode: Yes — Display the rulers. No — (Default) Do not display the rulers. You can view rulers in Preview mode in the DataWindow painter. Choose File>Print Preview, then File>Print Preview Rulers. However, the setting is not used at runtime. To see rulers during execution, set Print.Preview.Rulers in code.
Preview.Zoom	An integer indicating the zoom factor of the print preview. The default is 100%. You can view different zoom percentages in Preview mode in the DataWindow painter. Choose File>Print Preview, then File>Print Preview Zoom. However, the setting is not used at runtime. To change the zoom factor during execution, set Print.Preview.Zoom in code.

Property for Print	Value
PrinterName	A string containing the name of the printer you want to use to print the DataWindow report. If the printer name is not specified or if the named printer cannot be found at runtime, print output can be directed to the default printer for the user's machine by setting the CanUseDefaultPrinter property. Otherwise, an error is returned. Painter: Printer Name option.
Prompt	Whether a Printer Setup dialog displays before a job prints so the user can change the paper or other settings for the current printer. Values are: Yes — (Default) Display a Printer Setup dialog. No — Do not display a Printer Setup dialog. Choosing Cancel in the Printer Setup dialog dismisses the Setup dialog; it does not cancel printing. To allow the user to cancel printing, see the Print method. For DataStores, this property is ignored; a dialog is never displayed. Painter: Prompt Before Printing check box.
Quality	An integer indicating the quality of the output. Values are: 0 — Default 1 — High 2 — Medium 3 — Low 4 — Draft The user can specify the value in the system's Print dialog box if the printer driver supports it.
Scale	An integer specifying the scale of the printed output as a percent. The scaling percentage is passed to the print driver. If you have problems with scaling, you might be using a driver that does not support scaling. The user can specify the value in the system's Print dialog box if the printer driver supports it. For more information, see your print driver documentation.

Usage

In the painter Select the DataWindow by deselecting all controls; then set values in the Properties view, Print Specifications tab.

Examples

```
ls_data = dw_1.Object.DataWindow.Print.Scale

dw_1.Object.DataWindow.Print.Paper.Size = 3

ls_data = dw_1.Describe("DataWindow.Print.Scale")

dw_1.Modify("DataWindow.Print.Paper.Size = 3")

dw_1.Modify("DataWindow.Print.Margin.Top=500")
```

Printer

Description

The name of the printer for printing the DataWindow as specified in the system's printer selection dialog box.

PocketBuilder	✗
PowerBuilder	✓

Applies to

DataWindows

Syntax

PowerBuilder dot notation:

dw_control.Object.DataWindow.Printer = "*printername*"

Describe and Modify argument:

"DataWindow.Printer" { = *printername* }"

Processing

Description

The type of processing required to display the data in the selected presentation style.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

DataWindows

Syntax

PocketBuilder dot notation:

dw_control.Object.DataWindow.Processing

Describe argument:

"DataWindow.Processing"

Return values are:

0 — (Default) Form, group, query, or tabular
1 — Grid
2 — Label
3 — Graph
4 — Crosstab
5 — Composite
6 — OLE
7 — RichText

Examples

```
string setting
setting = dw_1.Object.DataWindow.Processing

setting = dw_1.Describe("DataWindow.Processing")
```

Protect

Description

The protection setting of a column. The Protect property overrides tab order settings. When a column is protected, the user cannot edit it even if the column's tab order is greater than 0.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

A column

Syntax

PocketBuilder dot notation:

dw_control.Object.*columnname*.Protect

Describe and Modify argument:

"*columnname*.Protect { = ' *integer* ' }"

Parameter	Description
columnname	The name of the column for which you want to get or set the protection.
integer	(exp) A boolean integer specifying whether the column is protected. Values are: 0 — False, the column is not protected. 1 — True, the column is protected. *Integer* can be a quoted DataWindow expression.

Usage

A user cannot change a column value if any one of these conditions are true:

- TabSequence is 0
- Edit.DisplayOnly is Yes when the column has the Edit edit style
- Protect is 1

Only the Protect property allows you to specify a conditional expression that makes some values in the column protected but not others.

In the painter Select the control and set the value in the Properties view, General tab (use a conditional expression).

Examples

```
string setting
setting = dw_1.Object.emp_stat.Protect

dw_1.Object.emp_stat.Protect=1

setting = dw_1.Describe("emp_stat.Protect")

dw_1.Modify("emp_stat.Protect=1")

dw_1.Modify("emp_stat.Protect='1~tIf(IsRowNew(),0,1)'")
```

QueryClear

Description

Removes the WHERE clause from a query. Note that the only valid setting is Yes.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

DataWindows

Syntax

PocketBuilder dot notation:

dw_control.Object.DataWindow.QueryClear

Modify argument:

"DataWindow.QueryClear { = *value* }"

Parameter	Description
value	Remove the WHERE clause from a query. Yes is the only valid value.

Examples

```
dw_1.Object.DataWindow.QueryClear = "yes"

dw_1.Modify("DataWindow.QueryClear=yes")
```

QueryMode

Description

Whether the DataWindow is in query mode. In query mode, the user can specify the desired data by entering WHERE criteria in one or more columns.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

DataWindow presentation styles
You cannot use QueryMode with DataWindow objects that use any of the following presentation styles: N-Up, Label, Crosstab, RichText, and Graph.

Applies to

DataWindows

Syntax

PocketBuilder dot notation:

dw_control.Object.DataWindow.QueryMode

Describe and Modify argument:

"DataWindow.QueryMode { = *value* }"

Parameter	Description
value	Whether the DataWindow is in query mode. Values are: Yes — Query mode is enabled. No — Query mode is disabled.

Usage

After the user specifies retrieval criteria in query mode, subsequent calls to Retrieve can use the new criteria. To retrieve data based on user selection, change the query mode back to No and use AcceptText to accept the user's specification before the next call to Retrieve.

Setting QuerySort to Yes also puts the DataWindow into query mode, changing the QueryMode property's value to Yes.

Query mode and secondary DataWindows When you are sharing data, you cannot turn on query mode for a secondary DataWindow. Trying to set the QueryMode or QuerySort properties results in an error.

Buffer manipulation and query mode A DataWindow *cannot* be in query mode when you call the RowsCopy function.

Examples

```
string setting
setting = dw_1.Object.DataWindow.QueryMode

dw_1.Object.DataWindow.QueryMode = "yes"
```

```
setting = dw_1.Describe("DataWindow.QueryMode")
dw_1.Modify("DataWindow.QueryMode=yes")
```

QuerySort

Description

Whether the result set is sorted when the DataWindow retrieves the data specified in query mode. When query sort is on, the user specifies sorting criteria in the first row of the query form.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

DataWindow presentation styles
You cannot use QuerySort with DataWindow objects that use any of the following presentation styles: N-Up, Label, Crosstab, RichText, and Graph.

Applies to

DataWindows

Syntax

PocketBuilder dot notation:

dw_control.Object.DataWindow.QuerySort

Describe and Modify argument:

"DataWindow.QuerySort { = *value* }"

Parameter	Description
value	Whether the data retrieved from query mode specifications is sorted. Values are: Yes — Sorting is enabled. No — Sorting is disabled.

Usage

If the DataWindow is not already in query mode, setting QuerySort to Yes also sets QueryMode to Yes, putting the DataWindow in query mode.

When you set QuerySort to No, the DataWindow remains in query mode until you also set QueryMode to No.

Query mode and secondary DataWindows When you are sharing data, you cannot turn on query mode for a secondary DataWindow. Trying to set the QueryMode or QuerySort properties results in an error.

Examples

```
string setting
setting = dw_1.Object.DataWindow.QuerySort

dw_1.Object.DataWindow.QuerySort = "yes"

setting = dw_1.Describe("DataWindow.QuerySort")

dw_1.Modify("DataWindow.QuerySort=yes")
```

RadioButtons.*property*

Description

Properties that control the appearance and behavior of a column with the RadioButton edit style.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Column controls

Syntax

PocketBuilder dot notation:

dw_control.Object.*columnname*.RadioButtons.*property*

Describe and Modify argument:

"*columnname*.RadioButtons.*property* { = *value* }"

Parameter	Description
columnname	The name of the column that has the RadioButton edit style.
property	A property for the RadioButton column. Properties and their settings are listed in the table below.
value	The value to be assigned to the property. For RadioButton properties, *value* cannot be a DataWindow expression.

Property for RadioButtons	Value
3D	Whether the radio buttons are 3D. Values are: Yes — Make the buttons 3D. No — Do not make the buttons 3D. Painter: 3D Look option.
Columns	An integer constant specifying the number of columns of radio buttons. Painter: Columns Across option.

Property for RadioButtons	Value
LeftText	Whether the text labels for the radio buttons are on the left side. Values are: Yes — The text is on the left of the radio buttons. No — The text is on the right of the radio buttons. Painter: Left Text option.
Scale	Whether the circle is scaled to the size of the font. Scale has an effect only when 3D is No. Values are: Yes — Scale the circles. No — Do not scale the circles. Painter: Scale Circles option.

Usage

In the painter Select the control and set the value in the Properties view, Edit tab when Style Type is RadioButtons.

Examples

```
string setting
setting = &
       dw_1.Object.emp_gender.RadioButtons.LeftText

dw_1.Object.emp_gender.RadioButtons.LeftText = "no"

setting = &
    dw_1.Describe("emp_gender.RadioButtons.LeftText")

dw_1.Modify("emp_gender.RadioButtons.LeftText=no")

dw_1.Modify("emp_gender.RadioButtons.3D=Yes")

dw_1.Modify("emp_gender.RadioButtons.Columns=2")
```

Range

Description

The rows in the DataWindow used in the graph control. Range can be all rows, the rows on the current page, or a group that you have defined for the DataWindow.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Graph controls

Syntax

PocketBuilder dot notation:

dw_control.Object.*controlname*.Range

Describe argument:

"*controlname*.Range"

Parameter	Description
controlname	The name of the graph control within the DataWindow that will display the graphed rows

Usage

Possible values are:

-1 — The rows on a single page in the DataWindow object
0 — All the rows in the DataWindow object
n — The number of a group level in the DataWindow object

In the painter Select the control and set the value in the Properties view, Data tab, Rows option.

Examples

```
string setting
setting = dw_1.Object.graph_salary.Range

setting = dw_1.Describe("graph_salary.Range")
```

ReadOnly

Description

Whether the DataWindow is read-only.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

DataWindows

Syntax

PocketBuilder dot notation:

dw_control.Object.DataWindow.ReadOnly

Describe and Modify argument:

"DataWindow.ReadOnly { = *value* }"

Parameter	Description
value	Whether the DataWindow is read-only. Values are: Yes — Make the DataWindow read-only. No — (Default) Do not make the DataWindow read-only.

Examples

```
string setting
setting = dw_1.Object.DataWindow.ReadOnly

dw_1.Object.DataWindow.ReadOnly="Yes"

setting = dw_1.Describe("DataWindow.ReadOnly")

dw_1.Modify("DataWindow.ReadOnly=Yes")
```

ReplaceTabWithSpace

Description

Whether tab characters embedded in the data for a DataWindow display as square boxes when the row is not the current row.

PocketBuilder	✗
PowerBuilder	✓

Applies to

DataWindows

Syntax

PocketBuilder dot notation:

dw_control.Object.DataWindow.ReplaceTabWithSpace

Describe and Modify argument:

"DataWindow.ReplaceTabWithSpace { = *value* }"

Parameter	Description
value	Whether tab characters embedded in the data for a DataWindow are replaced with spaces. Values are: Yes — Replace each tab character with four spaces. No — (Default) Do not replace tab characters.

Report

Description

Whether the DataWindow is a read-only report.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Style keywords

Syntax

SyntaxFromSQL:

Style (Report = *value*)

Parameter	Description
value	Whether the DataWindow is a read-only report, similar to a DataWindow created in the Report painter. Values are: Yes — The DataWindow is a read-only report. No — The DataWindow is not read-only.

Examples

```
SQLCA.SyntaxFromSQL(sqlstring, &
'Style(...Report = yes ...)', errstring)
```

ResetPageCount

Description

Specifies that a change in the value of the group column causes the page count to begin again at 0.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Group keywords

Syntax

SyntaxFromSQL:

Group (*col1* {*col2* ...} ... ResetPageCount)

Examples

```
SQLCA.SyntaxFromSQL(sql_syntax, &
       "Style(Type=Group) " &
           + "Group(#3 NewPage ResetPageCount)", &
               errorvar)
```

Resizeable

Description

Whether the user can resize the specified control.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Button, Column, Computed Field, Graph, GroupBox, Line, Oval, Picture, Rectangle, Report, RoundRectangle, TableBlob, and Text controls

Syntax

PocketBuilder dot notation:

dw_control.Object.*controlname*.Resizeable

Describe and Modify argument:

"*controlname*.Resizeable { = *value* }"

Parameter	Description
controlname	The control within the DataWindow whose Resizeable setting you want to get or set.
value	A boolean number indicating whether *controlname* can be resized. Values are: 0 — (False) The control cannot be resized. 1 — (True) The control can be resized.

Usage

In the painter Select the control and set the value in the Properties view, Position tab.

When you make the control resizable, set the Border property to the resizable border so the user knows it is resizable.

Examples

```
string setting
setting = dw_1.Object.graph_1.Resizeable

dw_1.Object.graph_1.Resizeable = 1

setting = dw_1.Describe("graph_1.Resizeable")

dw_1.Modify("graph_1.Resizeable=1")

dw_1.Modify("bitmap_1.Resizeable=0")
```

RetainNewLineChar

Description

Whether line feed and carriage return characters contained within a row in the DataWindow are converted to white space.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

DataWindows

Syntax

PocketBuilder dot notation:

dw_control.Object.DataWindow.RetainNewLineChar

Describe and Modify argument:

"DataWindow.RetainNewLineChar { = *value* }"

Parameter	Description
value	Whether line feed and carriage return characters embedded in the data for a DataWindow are replaced with white space. Values are: True — Line feed and carriage return characters within a row are retained. False — (Default) Line feed and carriage return characters within a row are converted to white space.

Examples

```
string str
str = dw_1.Object.DataWindow.RetainNewLineChar
dw_1.Object.DataWindow.RetainNewLineChar="False"
str = dw_1.Describe("DataWindow.RetainNewLineChar")
dw_1.Modify("DataWindow.RetainNewLineChar=False")
```

Retrieve

Description

The SQL statement for the DataWindow.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Retrieve is set in DataWindow syntax only for the Create function.

Applies to

Table keywords

Syntax

Table (... Retrieve = *selectstatement* ...)

Retrieve.AsNeeded

Description

Whether rows will be retrieved only as needed from the database. After the application calls the Retrieve function to get enough rows to fill the visible portion of the DataWindow, additional rows are "needed" when the user scrolls down to view rows that have not been viewed yet.

PocketBuilder	✗
PowerBuilder	✓

Applies to

DataWindows

Syntax

PowerBuilder dot notation:

dw_control.Object.DataWindow.Retrieve.AsNeeded

Describe and Modify argument:

"DataWindow.Retrieve.AsNeeded { = ' *value* ' }"

RichText.*property*

Description

Properties for the DataWindow RichText presentation style.

PocketBuilder	✗
PowerBuilder	✓

Applies to

DataWindows

Syntax

PowerBuilder dot notation:

dw_control.Object.DataWindow.RichText.*property*

Describe and Modify argument:

"DataWindow.RichText.*property* { = *value* }"

Rotation

Description

The degree of left-to-right rotation for the graph control within the DataWindow when the graph has a 3D type.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Graph controls

Syntax

PocketBuilder dot notation:

dw_control.Object.*graphname*.Rotation

Describe and Modify argument:

"*graphname*.Rotation = { ' *integer* ' }"

Parameter	Description
graphname	The name of the Graph control for which you want to get or set the rotation.
integer	(*exp*) The degree of rotation for the graph. Effective values range from -90 to 90. Integer can be a quoted DataWindow expression.

Usage

In the painter Select the control and set the value in the Properties view, Graph tab, Rotation scroll bar (enabled when a 3D graph type is selected).

Examples

```
string setting
setting = dw_1.Object.graph_1.Rotation

dw_1.Object.graph_1.Rotation=25

setting = dw_1.Describe("graph_1.Rotation")

dw_1.Modify("graph_1.Rotation=25")

dw_1.Modify("graph_1.Rotation='1~tHour(Now())'")
```

Row.Resize

Description

Whether the user can use the mouse to change the height of the rows in the detail area of the DataWindow.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

DataWindows

Syntax

PocketBuilder dot notation:

dw_control.Object.DataWindow.Row.Resize

Describe and Modify argument:

"DataWindow.Row.Resize { = *value* } "

Parameter	Description
value	Whether the user can resize the rows in the detail area. Values are: • 1 — Yes, the user can resize the rows. • 0 — No, the user cannot resize the rows.

Usage

In the painter Select the DataWindow by deselecting all controls; then set the value in the Properties view, General tab, Row Resize option (available when the presentation style is Grid or Crosstab).

Examples

```
string setting
setting = dw_1.Object.DataWindow.Row.Resize

dw_1.Object.DataWindow.Row.Resize = 0

setting = dw_1.Describe("DataWindow.Row.Resize")

dw_1.Modify("DataWindow.Row.Resize=0")
```

Rows_Per_Detail

Description

The number of rows in the detail area of an n-up DataWindow object. This property should be 1 unless the Type property for the Style keyword is Tabular.

PocketBuilder	✗
PowerBuilder	✓

Applies to

DataWindows

Syntax

PowerBuilder dot notation:

dw_control.Object.DataWindow.Rows_Per_Detail

Describe argument:

"DataWindow.Rows_Per_Detail"

SyntaxFromSQL:

DataWindow (... Rows_Per_Detail = *n* ...)

Selected

Description

A list of selected controls within the DataWindow.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

DataWindows

Syntax

PocketBuilder dot notation:

dw_control.Object.DataWindow.Selected

Describe and Modify argument:

"DataWindow.Selected = ' *list* ' "

Parameter	Description
list	A list of the controls you want to select. In the list you designate a group of controls by specifying a range of row numbers and a range of controls in the format: *startrow*/*endrow*/*startcontrol*/*endcontrol* To specify more than one group, separate each group with a semicolon: *startrow1*/*endrow1*/*startobj1*/*endobj1*;*startrow2*/*endrow2*/*startobj2*/*endobj2*;... Do not include spaces in the string. You must use column names, not column numbers.

Examples

```
setting = dw_1.Object.DataWindow.Selected

dw_1.Object.DataWindow.Selected = &
"1/10/emp_id/emp_name;12/23/salary/status"
```

```
setting = dw_1.Describe("DataWindow.Selected")

dw_1.Modify("DataWindow.Selected=" &
"'1/10/emp_id/emp_name;12/23/salary/status'")
```

Selected.Data

Description

A list describing the selected data in the DataWindow. Each column's data is separated by a tab and each row is on a separate line.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

DataWindows (Crosstab and Grid presentation styles only)

Syntax

PocketBuilder dot notation:

dw_control.Object.DataWindow.Selected.Data

Describe argument:

"DataWindow.Selected.Data"

Examples

```
string setting
setting = dw_1.Object.DataWindow.Selected.Data

setting = dw_1.Describe("DataWindow.Selected.Data")
```

Selected.Mouse

Description

Whether the user can use the mouse to select columns.

PocketBuilder	✗
PowerBuilder	✓

Applies to

DataWindows

Syntax

PowerBuilder dot notation:

dw_control.Object.DataWindow.Selected.Mouse

Describe and Modify argument:

"DataWindow.Selected.Mouse { = *value* }"

Series

See Axis, Axis.property, and DispAttr.fontproperty.

ShadeColor

Description

The color used for shading the back edge of the series markers when the graph's type is 3D. ShadeColor has no effect unless Series.ShadeBackEdge is 1 (Yes). If ShadeBackEdge is 0, the axis plane is the same color as the background color of the graph.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Graph controls

Syntax

PocketBuilder dot notation:

dw_control.Object.*graphname*.ShadeColor

Describe and Modify argument:

"*graphname*.ShadeColor { = ' *long* ' }"

Parameter	Description
graphname	The Graph control in the DataWindow for which you want to shade color.
long	(*exp*) A long number converted to a string specifying the color of the shading for axes of a 3D graph. You can use the RGB function in a DataWindow expression or in PowerScript to calculate the desired color value. However, be sure to convert the return value of the PowerScript function to a string. *Long* can be a quoted DataWindow expression.

Usage

To set the shade color for individual series markers, such as bars or pie slices, use the function SetDataStyle.

In the painter Select the control and set the value in the Properties view, General tab, Shade Color option.

Examples

```
string setting
setting = dw_1.Object.graph_1.ShadeColor

dw_1.Object.graph_1.ShadeColor = 16600000
```

```
setting = dw_1.Describe("graph_1.ShadeColor")

dw_1.Modify("graph_1.ShadeColor=16600000")

dw_1.Modify("graph_1.ShadeColor=" + &
       String(RGB(90,90,90)))

dw_1.Modify("graph_1.ShadeColor='0~t" &
       + If(salary>50000," &
       + String(RGB(100,90,90)) &
       + "," &
       + String(RGB(90,90,100)) &
       + ")'")
```

ShowDefinition

Description

Whether the DataWindow definition will display. The DataWindow will display the column names instead of data.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

DataWindows

Syntax

PocketBuilder dot notation:

dw_control.Object.DataWindow.ShowDefinition

Describe and Modify argument:

"DataWindow.ShowDefinition { = ' *value* ' }"

Parameter	Description
value	(*exp*) Whether the column names will display. Values are: • Yes — Display the column names. • No — Do not display the data, if any. *Value* can be a quoted DataWindow expression.

Examples

```
string setting
setting = dw_1.Object.DataWindow.ShowDefinition

dw_1.Object.DataWindow.ShowDefinition = "Yes"

setting = dw_1.Describe("DataWindow.ShowDefinition")

dw_1.Modify("DataWindow.ShowDefinition=Yes")
```

SizeToDisplay

Description

Whether the graph should be sized automatically to the display area.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Graph controls

Syntax

PocketBuilder dot notation:

dw_control.Object.*graphname*.SizeToDisplay

Describe and Modify argument:

"*graphname*.SizeToDisplay { = ' *value* ' }"

Parameter	Description
graphname	The graph control in the DataWindow for which you want to get or set adjustability.
value	(*exp*) A boolean number specifying whether to adjust the size of the graph to the display. Values are: • 0 — False, do not adjust the size of the graph. • 1 — True, adjust the size of the graph. *Value* can be a quoted DataWindow expression.

Usage

In the painter Select the control and set the value in the Properties view, General tab, Size To Display option.

Examples

```
string setting
setting = dw_1.Object.graph_1.SizeToDisplay

dw_1.Object.graph_1.SizeToDisplay = 0

setting = dw_1.Describe("graph_1.SizeToDisplay")

dw_1.Modify("graph_1.SizeToDisplay=0")
```

SlideLeft

Description

Whether the control moves to the left when other controls to the left leave empty space available. This property is for use with read-only controls and printed reports. It should not be used with data entry fields or controls.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Button, Column, Computed Field, Graph, GroupBox, Line, Oval, Picture, Rectangle, Report, RoundRectangle, TableBlob, and Text controls

Syntax

PocketBuilder dot notation:

dw_control.Object.*controlname*.SlideLeft

Describe and Modify argument:

"*controlname*.SlideLeft { = ' *value* ' }"

Parameter	Description
controlname	The name of the control for which you want to get or set the Slide setting.
value	(*exp*) Whether the control slides left when there is empty space to its left. Values are: • Yes — The control will slide left into available space. • No — The control will remain in position. *Value* can be a quoted DataWindow expression.

Usage

In the painter Select the control and set the value in the Properties view, Position tab, Slide Left check box.

Examples

```
string setting
setting = dw_1.Object.graph_1.SlideLeft

dw_1.Object.emp_lname.SlideLeft = "yes"

setting = dw_1.Describe("graph_1.SlideLeft")

dw_1.Modify("emp_lname.SlideLeft=yes")
```

SlideUp

Description

Whether the control moves up when other controls above it leave empty space available. This property is for use with read-only controls and printed reports. It should not be used with data entry fields or controls.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Button, Column, Computed Field, Graph, GroupBox, Line, Oval, Picture, Rectangle, Report, RoundRectangle, TableBlob, and Text controls

Syntax

PocketBuilder dot notation:

dw_control.Object.*controlname*.SlideUp

Describe and Modify argument:

"*controlname*.SlideUp { = ' *value* ' }"

Parameter	Description
controlname	The name of the control for which you want to get or set the Slide setting.
value	(*exp*) How the control slides up when there is empty space above it. Values are: • AllAbove — Slide the control up if all the controls in the row above it are empty. • DirectlyAbove — Slide the column or control up if the controls directly above it are empty. • No — The control will not slide up. *Value* can be a quoted DataWindow expression.

Usage

In the painter Select the control and set the value in the Properties view, Position tab, Slide Up check box.

Examples

```
string setting
setting = dw_1.Object.graph_1.SlideUp

dw_1.Object.emp_lname.SlideUp = "no"

setting = dw_1.Describe("graph_1.SlideUp")

dw_1.Modify("emp_lname.SlideUp=no")
```

Sort

Description

Sort criteria for a newly created DataWindow. To specify sorting for existing DataWindows, see the SetSort and Sort functions.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Table keywords in DataWindow syntax

Syntax

DataWindow syntax for Create function:

Table (... Sort = *stringexpression* ...)

Parameter	Description
stringexpression	A string whose value represents valid sort criteria. See the SetSort function for the format for sort criteria. If the criteria string is NULL, PocketBuilder prompts for a sort specification when it displays the DataWindow.

Spacing

Description

The gap between categories in a graph.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Graph controls

Syntax

PocketBuilder dot notation:

dw_control.Object.*graphname*.Spacing

Describe and Modify argument:

"*graphname*.Spacing { = ' *integer* ' }"

Parameter	Description
graphname	The name of the graph control in the DataWindow for which you want to get or set the spacing.

Parameter	Description
integer	(*exp*) An integer specifying the gap between categories in the graph. You specify the value as a percentage of the width of the data marker. For example, in a bar graph, 100 is the width of one bar; 50 is half a bar, and so on. *Integer* can be a DataWindow expression.

Usage

In the painter Select the control and set the value in the Properties view, General tab, Spacing option.

Examples

```
string setting
setting = dw_1.Object.graph_1.Spacing

dw_1.Object.graph_1.Spacing = 120

setting = dw_1.Describe("graph_1.Spacing")

dw_1.Modify("graph_1.Spacing=120")
```

Sparse

Description

The names of repeating columns that will be suppressed in the DataWindow.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

DataWindows

Syntax

PocketBuilder dot notation:

dw_control.Object.DataWindow.Sparse

Describe and Modify argument:

"DataWindow.Sparse { = ' *list* ' }"

Parameter	Description
list	(*exp*) A tab-separated list of column names to be suppressed. *List* can be a quoted DataWindow expression.

Create function (include at the end of the DataWindow syntax):

Sparse (names = "*col1*~t*col2*~t*col3 ...*")

Usage

In the painter Set the value using Rows>Suppress Repeating Values.

Examples

```
string setting
```

```
setting = dw_1.Object.DataWindow.Sparse
dw_1.Object.DataWindow.Sparse = 'col1~tcol2'
setting = dw_1.Describe("DataWindow.Sparse")
dw_1.Modify("DataWindow.Sparse='col1~tcol2'")
```

Storage

Description

The amount of virtual storage in bytes that has been allocated for the DataWindow object.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

DataWindows

Syntax

PocketBuilder dot notation:

dw_control.Object.DataWindow.Storage

Describe argument:

"DataWindow.Storage"

Usage

Canceling a query that uses too much storage You can check this property in the script for the RetrieveRow event in the DataWindow control and cancel a query if it is consuming too much storage.

Examples

```
string setting
setting = dw_1.Object.DataWindow.Storage
setting = dw_1.Describe("DataWindow.Storage")
IF Long(setting) > 50000 THEN RETURN 1
```

StoragePageSize

Description

The default page size for DataWindow storage.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

DataWindows

Syntax

PocketBuilder dot notation:

dw_control.Object.DataWindow.StoragePageSize

Describe and Modify argument:

"DataWindow.StoragePageSize { = ' *size* ' }"

Parameter	Description
size	Two values are provided to enable the DataWindow to use the available virtual memory most efficiently in the current environment: • LARGE (Recommended) • MEDIUM

Usage

Set this property to avoid out of memory errors when performing large retrieve, import, or RowsCopy operations. The property must be set *before* the operation is invoked.

Examples

```
dw_1.Modify("datawindow.storagepagesize='LARGE'")
dw_1.object.datawindow.storagepagesize='large'
```

Summary.*property*

See Bandname.property.

SuppressEventProcessing

Description

Whether the ButtonClicked or ButtonClicking event is fired for this particular button.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Button controls

Syntax

PocketBuilder dot notation:

dw_control.Object.*buttonname*.SuppressEventProcessing

Describe and Modify argument:

"*buttonname*.SuppressEventProcessing { = ' *value* ' }"

Parameter	Description
buttonname	The name of the button control for which you want to suppress event processing.
value	Whether event processing is to occur. Values are: Yes — The event should be fired. No — The event should not be fired.

Usage

In the painter Select the control and set the value in the Properties view, General tab.

Examples

```
string setting
dw_1.Object.b_name.SuppressEventProcessing = "Yes"

setting = &
  dw_1.Describe("b_name.SuppressEventProcessing")

dw_1.Modify("b_name.SuppressEventProcessing ='No'")
```

Syntax

Description

The complete syntax for the DataWindow.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

DataWindows

Syntax

PocketBuilder dot notation:

dw_control.Object.DataWindow.Syntax

Describe argument:

"DataWindow.Syntax"

Examples

```
setting = dw_1.Object.DataWindow.Syntax

setting = dw_1.Describe("DataWindow.Syntax")
```

Syntax.Data

Description

The data in the DataWindow object described in parse format (the format required by the DataWindow parser).

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

DataWindows

Syntax

PocketBuilder dot notation:

dw_control.Object.DataWindow.Syntax.Data

Describe argument:

"DataWindow.Syntax.Data"

Usage

Use this property with the Syntax property to obtain the description of the DataWindow object and the data. Using this information, you can create a syntax file that represents both the structure and data of a DataWindow at an instant in time. You can then use the syntax file as a DropDownDataWindow containing redefined data at a single location or to mail this as a text object.

Syntax.Modified

Description

Whether the DataWindow syntax has been modified by a function call or user intervention. Calling the Modify, SetSort, or SetFilter function or changing the size of the DataWindow grid automatically sets Syntax.Modified to Yes.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

DataWindows

Syntax

PocketBuilder dot notation:

dw_control.Object.DataWindow.Syntax.Modified

Describe and Modify argument:

"DataWindow.Syntax.Modified { = *value* }"

Parameter	Description
value	Whether the DataWindow syntax has been modified. Values are: • Yes — DataWindow syntax has been modified. • No — DataWindow has not been modified.

Usage

Use this property in Modify to set Syntax.Modified to No after you cause a change in the syntax that does not affect the user (such as setting preview on).

Examples

```
string setting
setting = dw_1.Object.DataWindow.Syntax.Modified

dw_1.Object.DataWindow.Syntax.Modified = "No"

setting = dw_1.Describe("DataWindow.Syntax.Modified")

dw_1.Modify("DataWindow.Syntax.Modified=No")
```

Table (for Create)

Description

The section of the DataWindow syntax that specifies information about the DataWindow's database table, including the name of the update table.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Use Table in DataWindow syntax for the Create function.

Syntax

Does not apply.

Usage

Use this property to redefine a DataWindow result set. You can add a column, change the datatype of a column, or make other changes to the table section of your DataWindow involving properties that are not accessible through Modify calls or dot notation.

Caution
When you use this property to redefine the result set, you must redefine the table section in its entirety.

You can call the GetItem and SetItem functions to access columns added using this property, but the columns do not display in the DataWindow unless you call Modify("create column(...)") to add them.

To redefine your table section:

1 Export your DataWindow object to a DOS file.

2 Copy only the table section into your script.

3 Modify the table section to meet your needs.

4 Put the new table definition into a string variable. Change existing double quotation marks (") in the string to single quotation marks (') and change the tilde quotation marks to tilde tilde single quotation marks (~~').

5 Call Modify. Modifying the table section of your DataWindow causes the DataWindow to be reset.

6 (Optionally) Call Modify to add the column to the DataWindow display.

Table (for TableBlobs)

Description

The name of the database table that contains the blob.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

TableBlob controls

Syntax

PocketBuilder dot notation:

dw_control.Object.*tblobname*.Table

Describe and Modify argument:

"*tblobname*.Table { = ' *tablename* ' }"

Parameter	Description
tblobname	The name of the TableBlob control in the DataWindow.
tablename	(*exp*) A string specifying the name of the table that contains the blob data. *Tablename* can be a quoted DataWindow expression.

Usage

In the painter Select the control and set the value in the Properties view, Definition tab, Table option.

Examples

```
setting = dw_1.Object.blob_1.Table

dw_1.Object.blob_1.Table = "emp_pictures"
```

```
setting = dw_1.Describe("blob_1.Table")
dw_1.Modify("blob_1.Table='emp_pictures'")
```

Table.*property*

Description

Properties for the DataWindow's DBMS connection.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

You can also specify stored procedures for update activities. For information, see Table.*sqlaction.property*.

Applies to

DataWindows

Syntax

PocketBuilder dot notation:

dw_control.Object.DataWindow.Table.*property*

Describe and Modify argument:

"DataWindow.Table.*property* { = *value* }"

Parameter	Description
property	A property for the DataWindow's DBMS connection. Properties and appropriate values are listed in the table below.
value	The value to be assigned to the property.

Property for Table	Value
Arguments	(Read only) A string containing retrieval argument names and types for the DataWindow.
CrosstabData	A string containing a tab-separated list of the expressions used to calculate the values of columns in a crosstab DataWindow.
Delete.Argument	(Internal use only) A string containing arguments to pass to the delete method.
Delete.Method	(Internal use only) The name of the method.
Delete.Type	(Internal use only) Currently stored procedure is the only type implemented.

Property for Table	Value
Data.Storage	A string indicating whether table data is to be kept in memory or offloaded to disk. Values are: • Memory (Default) — Table data is to be kept in memory. • Disk — Table data is to be offloaded to disk. Painter: Rows>Retrieve Options>Rows to Disk.
Filter	(*exp*) A string containing the filter for the DataWindow. Filters are expressions that can evaluate to TRUE or FALSE. The Table.Filter property filters the data before it is retrieved. To filter data already in the DataWindow's buffers, use the Filter property or the SetFilter and Filter functions. The filter string can be a quoted DataWindow expression. Painter: Rows>Filter.
GridColumns	(Read-only) The grid columns of a DataWindow.
Insert.Argument	(Internal use only) A string containing arguments to pass to the insert method.
Insert.Method	(Internal use only) The name of the method.
Insert.Type	(Internal use only) Currently stored procedure is the only type implemented.
Procedure	A string that contains the number of the result set returned by the stored procedure to populate the DataWindow object. You can use this property only if your DBMS supports stored procedures. Use this property to change the stored procedure or to change the data source from a SELECT statement or script to a stored procedure (see the example). Painter: Set when Stored Procedure is selected as a data source.

Property for Table	Value
Select	A string containing the SQL SELECT statement that is the data source for the DataWindow. Use this property to specify a new SELECT statement or change the data source from a stored procedure or Script to a SELECT statement. Table.Select has several advantages over the SetSQLSelect function: • It is faster. PocketBuilder does not validate the statement until retrieval. • You can change data source for the DataWindow. For example, you can change from a SELECT to a Stored Procedure. • You can use none or any of the arguments defined for the DataWindow object in the SELECT. You cannot use arguments that were not previously defined for the DataWindow object. Describe always tries to return a SQL SELECT statement. If the database is not connected and the property's value is a PBSELECT statement, Describe will convert it to a SQL SELECT statement if a SetTransObject function has been executed for the DataWindow object. If you are using describeless retrieval (the StaticBind DBParm parameter is set to 1), you cannot use the Select property. Painter: Set when Select or Quick Select is selected as a data source.
Select.Attribute	(Read-only) A string containing the PBSELECT statement for the DataWindow.
Sort	(*exp*) A string containing the sort criteria for the DataWindow—for example, "1A,2D" (column 1 ascending, column 2 descending). The Table.Sort property sorts the data before it is retrieved. To sort data already in the DataWindow's buffers, use the SetSort and Sort functions. The value for Sort is quoted and can be a DataWindow expression. Painter: Rows>Sort.
SQLSelect	The most recently executed SELECT statement. Setting this has no effect. See Select in this table.
Update.Argument	(Internal use only) A string containing arguments to pass to the update method.
Update.Method	(Internal use only) The name of the method.

Property for Table	Value
Update.Type	(Internal use only) Currently stored procedure is the only type implemented.
UpdateKey InPlace	Whether the key column can be updated in place or the row has to be deleted and reinserted. This value determines the syntax PocketBuilder generates when a user modifies a key field: • Yes — Use the UPDATE statement when the key is changed so that the key is updated in place. • No — Use a DELETE and an INSERT statement when the key is changed. **Caution** When there are multiple rows in a DataWindow object and the user switches keys or rows, updating in place might fail due to DBMS duplicate restrictions. Painter: Rows>Update Properties, Key Modification.
UpdateTable	A string specifying the name of the database table used to build the Update syntax. Painter: Rows>Update Properties, Table to Update.
UpdateWhere	An integer indicating which columns will be included in the WHERE clause of the Update statement. The value of UpdateWhere can impact performance or cause lost data when more than one user accesses the same tables at the same time. Values are: • 0 — Key columns only (risk of overwriting another user's changes, but fast). • 1 —Key columns and all updatable columns (risk of preventing valid updates; slow because SELECT statement is longer). • 2 — Key and modified columns (allows more valid updates than 1 and is faster but not as fast as 0). For more about the effects of this setting, see the discussion of the Specify Update Characteristics dialog box in the *Users Guide*. Painter: Rows>Update Properties, Where Clause for Update/Delete.

Examples

```
setting = dw_1.Object.DataWindow.Table.Sort

dw_1.Object.DataWindow.Table.Data.Storage &
       = "disk"
```

```
dw_1.Object.DataWindow.Table.Filter = "salary>50000"

setting = dw_1.Describe("DataWindow.Table.Sort")

dw_1.Modify("DataWindow.Table.Filter='salary>50000'")

dw_1.Modify &
       (" DataWindow.Table.Procedure= &
           '1 Execute MyOwner MyProcName;1 &
              @NameOfProcArg=:NameOfDWArg,
@NameOfProcArg=:NameOfDWArg...' ")

sqlvar = 'SELECT ... WHERE ...'
dw_1.Modify("DataWindow.Table.Select='" + sqlvar + "'")
```

Table.*sqlaction.property*

Description

The way data is updated in the database. When the Update method is executed, it can send UPDATE, INSERT, and DELETE SQL statements to the DBMS. You can specify that a stored procedure be used instead of the default SQL statement for each type of data modification.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

DataWindows

Syntax

PocketBuilder dot notation:

dw_control.Object.DataWindow.Table.*sqlaction*.*property*

Describe and Modify argument:

"DataWindow.Table.*sqlaction*.*property* { = *value* }"

Parameter	Description
sqlaction	The SQL statement that would ordinarily be executed as part of a database update. Values are: • UPDATE • INSERT • DELETE
property	A property for *sqlaction*. Properties and appropriate values are listed in the table below.
value	The value to be assigned to the property.

Property for Table	Value
Arguments	A string specifying the arguments used in the stored procedure. The string takes this format: ("*argname*", *valuetype* { =("*valuesrc*" {, *datasrc*, *paramtype* }) *Argname* is the name of the stored procedure parameter. *Valuetype* is one of the keywords described below. *Datasrc* and *paramtype* apply to the COLUMN keyword. *Valuesrc* is the column, computed field, or expression that produces the value to be passed to the stored procedure.
Method	A string specifying the name of the stored procedure. The stored procedure is used only if the value of Type is SP.
Type	Specifies whether the database update is performed using a stored procedure. Values are: • **SP** The update is performed using a stored procedure. • **SQL** The update is performed using standard SQL syntax (default).

Keyword for valuetype	Description
COLUMN	The argument value will be taken from the table and column named in *valuesrc*. *Valuesrc* has the form: "*tablename.column*" For COLUMN, you must also specify whether the data is the new or original column value. Values for *datasrc* are: • **NEW** The new column value that is being sent to the database. • **ORIG** The value that the DataWindow originally read from the database. You can also specify the type of stored procedure parameter. Values for *paramtype* are: • **IN** (Default) An input parameter for the procedure. • **OUT** An output parameter for the procedure. The DataWindow will assign the resulting value to the current row and column (usually used for identity and timestamp columns). • **INOUT** An input and output parameter. A sample string for providing a column argument is: `("empid", COLUMN=("employee.empid", ORIG, IN))`

Keyword for valuetype	Description
COMPUTE	The computed field named in *valuesrc* is the source of the value passed to the stored procedure. A sample string for providing a computed field argument is: `("newsalary", COMPUTE=("salary_calc"))`
EXPRESSION	The expression specified in *valuesrc* is evaluated and passed to the stored procedure. A sample string for providing an expression argument is: `("dept_name", EXPRESSION=("LookUpDisplay(dept_id)"))`
UNUSED	No value is passed to the stored procedure.

Usage

In the painter Set the values using Rows>Stored Procedure Update. Select the tab page for the SQL command you want to associate with a stored procedure.

In code If you enable a DataWindow object to use stored procedures to update the database when it is not already using stored procedures, you must change Type to SP first. Setting Type ensures that internal structures are built before you set Method and Arguments. If you do not change Type to SP, then setting Method or Arguments will fail.

When the values you specify in code are nested in a longer string, you must use the appropriate escape characters for quotation marks.

Examples

Each is all on one line:

```
dw_x.Describe("DataWindow.Table.Delete.Method")

dw_x.Describe("DataWindow.Table.Delete.Arguments")

dw_x.Modify("DataWindow.Table.Delete.Type=SP")

dw_x.Modify("DataWindow.Table.Delete.Arguments= &
((~"id~", COLUMN=(~"department.dept_id!~", ORIG)))")

dw_x.Modify("DataWindow.Table.Delete.Method= &
 ~"spname~"")
```

TabSequence

Description

The number assigned to the specified control in the DataWindow's tab order. You can also call the SetTabOrder function to change TabSequence.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Column controls

Syntax

PocketBuilder dot notation:

dw_control.Object.*columnname.*TabSequence

Describe and Modify argument:

"*columnname.*TabSequence { = *number* }"

Parameter	Description
columnname	The name of the column whose tab order you want to get or set.
number	A number from 0 to 32000 specifying the position of the column in the tab order. A value of 0 takes the column out of the tab order and makes it read-only.

Usage

In the painter Set the value using Format>Tab Order.

Tab order changes have no effect in grid DataWindow objects
In a grid DataWindow object, the tab sequence is always left to right (except on right-to-left operating systems). Changing the tab value to any number other than 0 has no effect.

Examples

```
string setting
setting = dw_1.Object.emp_name.TabSequence

dw_1.Object.emp_name.TabSequence = 10

setting = dw_1.Describe("emp_name.TabSequence")

dw_1.Modify("emp_name.TabSequence = 10")
```

Tag

Description

The tag value of the specified control. The tag value can be any text you see fit to use in your application.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Button, Column, Computed Field, Graph, GroupBox, Oval, Picture, Rectangle, Report, RoundRectangle, TableBlob, and Text controls

Syntax

PocketBuilder dot notation:

dw_control.Object.*controlname*.Tag

Describe and Modify argument:

"*controlname*.Tag { = ' *string* ' }"

Parameter	Description
controlname	The name of a control in the DataWindow.
string	(*exp*) A string specifying the tag for *controlname*. *String* is quoted and can be a DataWindow expression.

Usage

In the painter Select the control and set the value in the Properties view, General tab, Tag option.

Examples

```
setting = dw_1.Object.blob_1.Tag

dw_1.Object.graph_1.Tag = 'Graph of results'

setting = dw_1.Describe("blob_1.Tag")

dw_1.Modify("graph_1.Tag = 'Graph of results'")
```

Target

Description

The columns and expressions whose data is transferred from the DataWindow to the OLE object.

PocketBuilder	✗
PowerBuilder	✓

Applies to

OLE Object controls

Syntax

PowerBuilder dot notation:

dw_control.Object.*oleobjectname*.Target

Describe and Modify argument:

"*oleobjectname*.Target { = ' *columnlist* ' }"

Template

Description

The name of a file that will be used to start the application in OLE.

PocketBuilder	✗
PowerBuilder	✓

Applies to

TableBlob controls

Syntax

PowerBuilder dot notation:

dw_control.Object.*tblobname*.Template

Describe and Modify argument:

"*tblobname*.Template { = ' *string* ' }"

Text

Description

The text of the specified control.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Button, GroupBox, and Text controls

Syntax

PocketBuilder dot notation:

dw_control.Object.*textname*.Text

Describe and Modify argument:

"*textname*.Text { = ' *string* ' }"

Parameter	Description
textname	The name of a control in the DataWindow.

Parameter	Description
string	(*exp*) A string specifying the text for *textname*. To specify an accelerator key in the text, include an ampersand before the desired letter. The letter will display underlined. *String* is quoted and can be a DataWindow expression.

Usage

In the painter Select the control and set the value in the Properties view, General tab, Text option.

Examples

```
setting = dw_1.Object.text_1.Text

dw_1.Object.text_1.Text = "Employee &Name"

setting = dw_1.Describe("text_1.Text")

dw_1.Modify("text_1.Text='Employee &Name'")
```

Timer_Interval

Description

The number of milliseconds between the internal timer events. When you use time in a DataWindow, an internal timer event is triggered at the interval specified by Timer_Interval. This determines how often time fields are updated.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

DataWindows

Syntax

PocketBuilder dot notation:

dw_control.Object.DataWindow.Timer_Interval

Describe and Modify argument:

"DataWindow.Timer_Interval { = *number* }"

SyntaxFromSQL:

DataWindow (Timer_Interval = *number*)

Parameter	Description
number	An integer specifying the interval between timer events in milliseconds. The default is 60,000 milliseconds or one minute.

Usage

In the painter Select the DataWindow by deselecting all controls; then set the value in the Properties view, General tab, Timer Interval option.

Examples

```
string setting
setting = dw_1.Object.DataWindow.Timer_Interval

dw_1.Object.DataWindow.Timer_Interval = 10000

setting = dw_1.Describe("DataWindow.Timer_Interval")

dw_1.Modify("DataWindow.Timer_Interval=10000")
```

Title

Description

The title of the graph.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Graph controls

Syntax

PocketBuilder dot notation:

dw_control.Object.*graphname*.Title

Describe and Modify argument:

"*graphname*.Title { = ' *titlestring* ' }"

Parameter	Description
graphname	In the DataWindow object, the name of the Graph control for which you want to get or set the title.
titlestring	A string specifying the graph's title.

Usage

In the painter Select the control and set the value in the Properties view, General tab, Title option.

The default expression for the Title.DispAttr.DisplayExpression property is "title", which refers to the value of the Title property. The display expression can combine the fixed text of the Title property with other text, functions, and operators. If the expression for Title.DispAttr.DisplayExpression does not include the Title property, then the value of the Title property will be ignored.

For an example, see DispAttr.fontproperty.

Examples

```
setting = dw_1.Object.gr_1.Title
```

```
dw_1.Object.gr_1.Title = 'Sales Graph'
setting = dw_1.Describe("gr_1.Title")
dw_1.Modify("gr_1.Title = 'Sales Graph'")
```

Title.DispAttr.*fontproperty*

See DispAttr.fontproperty.

Trail_Footer

Description

Whether the footer of a nested report is displayed at the end of the report or at the bottom of the page. Trail_Footer applies only to reports in a composite DataWindow. Setting Trail_Footer to No forces controls following the report onto a new page.

PocketBuilder	✗
PowerBuilder	✓

Applies to

Report controls

Syntax

PowerBuilder dot notation:

dw_control.Object.*reportname*.Trail_Footer

Describe and Modify argument:

"*reportname*.Trail_Footer { = *value* }"

Trailer.#.*property*

See Bandname.property.

Type

Description

The type of the control (for Describe) or the type of presentation style (for SyntaxFromSQL).

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

PocketBuilder dot notation:

dw_control.Object.*controlname*.Type

Describe argument:

"*controlname*.Type"

Parameter	Description
controlname	The name of the control for which you want the type. Valid values are: • datawindow • bitmap (for Picture) • button • column • compute (for Computed Field) • graph • groupbox • line • ellipse (for Oval) • rectangle • report • roundrectangle • tableblob • text

SyntaxFromSQL:

Style (Type = *value*)

Parameter	Description
value	A keyword specifying the presentation style for the DataWindow object. Keywords are: • (Default) Tabular • Grid • Form (for the Freeform style) • Graph • Group

Examples

```
string setting
setting = dw_1.Object.emp_name.Type

setting = dw_1.Describe("emp_name.Type")

SQLCA.SyntaxFromSQL(sqlstring, &
'Style(... Type=grid ...)', errstring)
```

Units

Description

The unit of measure used to specify measurements in the DataWindow object. You set this in the DataWindow Style dialog box when you define the DataWindow object.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

DataWindows

Syntax

PocketBuilder dot notation:

dw_control.Object.DataWindow.Units

Describe argument:

"DataWindow.Units"

SyntaxFromSQL:

DataWindow (Units = *value*)

Parameter	Description
value	The type of units for measurements in the DataWindow. Values are: 0 — PowerBuilder units 1 — Display pixels 2 — 1/1000 of a logical inch 3 — 1/1000 of a logical centimeter

Usage

PowerBuilder units and display pixels are adjusted for printing.

In the painter Select the DataWindow by deselecting all controls; then set the value in the Properties view, General tab, Units option.

Examples

```
string setting
setting = dw_1.Object.DataWindow.Units

setting = dw_1.Describe("DataWindow.Units")
```

Update

Description

Whether the specified column is updatable. Each updatable column is included in the SQL statement that the Update function sends to the database. All updatable columns should be in the same database table.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Column controls

Syntax

PocketBuilder dot notation:

dw_control.Object.*columnname*.Update

Describe and Modify argument:

"*columnname*.Update { = *value* }"

Parameter	Description
columnname	The column for which you want to get or set the updatable status
value	Whether the column is updatable. Values are: Yes — Include the column in the SQL statement for updating the database. No — Do not include the column in the SQL statement.

Usage

In the painter Set the value using Rows>Update Properties, Updatable Columns option

Examples

```
string setting
setting = dw_1.Object.emp_name.Update

dw_1.Object.emp_name.Update = "No"

setting = dw_1.Describe("emp_name.Update")

dw_1.Modify("emp_name.Update=No")
```

Validation

Description

The validation expression for the specified column. Validation expressions are expressions that evaluate to TRUE or FALSE. They provide checking of data that the user enters in the DataWindow.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

To set the validation expression, you can also use the SetValidate function. To check the current validation expression, use the GetValidate function.

Applies to

Column controls

Syntax

PocketBuilder dot notation:

dw_control.Object.*columnname*.Validation

Describe and Modify argument:

"*columnname*.Validation { = ' *validationstring* ' }"

Parameter	Description
columnname	The column for which you want to get or set the validation rule.
validationstring	(*exp*) A string containing the rule that will be used to validate data entered in the column. Validation rules are expressions that evaluate to TRUE or FALSE. *Validationstring* is quoted and can be a DataWindow expression.

Usage

In the painter Set the value using the Column Specifications view, Validation Expression option.

Use operators, functions, and columns to build an expression. Use Verify to test it.

Examples

```
string setting
setting = dw_1.Object.emp_status.Validation

setting = dw_1.Describe("emp_status.Validation")
```

ValidationMsg

Description

The message that PocketBuilder displays instead of the default message when an ItemError event occurs in the column.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Column controls

Syntax

PocketBuilder dot notation:

dw_control.Object.*columnname*.ValidationMsg

Describe and Modify argument:

"*columnname*.ValidationMsg { = ' *string* ' }"

Parameter	Description
columnname	The column for which you want to get or set the error message displayed when validation fails.
string	(*exp*) A string specifying the error message you want to set. *String* is quoted and can be a DataWindow expression.

Usage

In the painter Set the value using the Column Specifications view, Validation Message option.

Examples

```
string setting
setting = dw_1.Object.emp_salary.ValidationMsg

dw_1.Object.emp_salary.ValidationMsg = &
"Salary must be between 10,000 and 100,000"

setting = dw_1.Describe("emp_salary.ValidationMsg")

dw_1.Modify("emp_salary.ValidationMsg = " &
"'Salary must be between 10,000 and 100,000'")
```

Values (for columns)

Description

The values in the code table for the column.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Column controls

Syntax

PocketBuilder dot notation:

dw_control.Object.*columnname*.Values

Describe and Modify argument:

"*columnname*.Values { = ' *string* ' }"

Parameter	Description
columnname	The column for which you want to specify the contents of the code table.
string	(*exp*) A string containing the code table values for the column. In the string, separate the display values and the actual values with a tab character, and separate multiple pairs of values with a slash using this format: "*displayval*~t*actualval*/*displayval*~t*actualval*/ ..." For example: `"red~t1/white~t2"` *String* is quoted and can be a DataWindow expression.

Usage

In the painter Select the control and set the value in the Properties view, Edit tab.

When Style Type is *DropDownListBox*, fill in the Display Value and Data Value columns for Code Table.

When Style is *Edit* or *EditMask*, select the Use Code Table or Code Table check box and fill in the Display Value and Data Value columns for Code Table.

Examples

```
setting = dw_1.Object.emp_status.Values

dw_1.Object.emp_status.Values = &
       "Active~tA/Part Time~tP/Terminated~tT"

setting = dw_1.Describe("emp_status.Values")

dw_1.Modify("emp_status.Values=" &
       + "'Active~tA/Part Time~tP/Terminated~tT'")
```

Values (for graphs)

See Axis, Axis.property, and DispAttr.fontproperty.

Vertical_Size

Description

The height of the columns in the detail area of the DataWindow object. Vertical_Size is meaningful only when Type is Form (meaning the Freeform style). When a column reaches the specified height, PocketBuilder starts a new column to the right of the current column. The space between columns is specified in the Vertical_Spread property.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Style keywords

Syntax

SyntaxFromSQL:

Style (Vertical_Size = *value*)

Parameter	Description
value	An integer specifying the height of the columns in the detail area of the DataWindow object area in the units specified for the DataWindow.

Examples

```
SQLCA.SyntaxFromSQL(sqlstring, &
'Style(... Vertical_Size=1225...)', errstring)
```

Vertical_Spread

Description

The vertical space between columns in the detail area of the DataWindow object. Vertical_Spread is meaningful only when Type is Form (meaning the Freeform style). The Vertical_Size property determines when to start a new column.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Style keywords

Syntax

SyntaxFromSQL:

Style (Vertical_Spread = *value*)

Parameter	Description
value	An integer specifying the vertical space between columns in the detail area of the DataWindow object area in the units specified for the DataWindow.

Examples

```
SQLCA.SyntaxFromSQL(sqlstring, &
       'Style(... Vertical_Spread=25...)', errstring)
```

VerticalScrollMaximum

Description

The maximum height of the scroll box of the DataWindow's vertical scroll bar. This value is set by PocketBuilder based on the content of the DataWindow. Use VerticalScrollMaximum with VerticalScrollPosition to synchronize vertical scrolling in multiple DataWindow objects. The value is a long.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

DataWindows

Syntax

PocketBuilder dot notation:

dw_control.Object.DataWindow.VerticalScrollMaximum

Describe argument:

"DataWindow.VerticalScrollMaximum"

Examples

```
string setting
setting = dw_1.Object.DataWindow.VerticalScrollMaximum

setting = &

dw_1.Describe("DataWindow.VerticalScrollMaximum")
```

VerticalScrollPosition

Description

The position of the scroll box in the vertical scroll bar. Use VerticalScrollMaximum with VerticalScrollPosition to synchronize vertical scrolling in multiple DataWindow objects.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

DataWindows

Syntax

PocketBuilder dot notation:

dw_control.Object.DataWindow.VerticalScrollPosition

Describe and Modify argument:

"DataWindow.VerticalScrollPosition { = *scrollvalue* }"

Parameter	Description
scrollvalue	A long specifying the position of the scroll box in the vertical scroll bar of the DataWindow.

Examples

```
string spos1
spos1 = dw_1.Object.DataWindow.VerticalScrollPosition
```

```
string spos1, smax, sscroll, modstring
spos1 = &
    dw_1.Describe("DataWindow.VerticalScrollPosition")
smax = &
    dw_1.Describe("DataWindow.VerticalScrollMaximum")
sscroll = String(Long(smax)/2)
modstring = &
       "DataWindow.VerticalScrollPosition=" + sscroll
dw_1.Modify(modstring)
```

Visible

Description

Whether the specified control in the DataWindow is visible.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Button, Column, Computed Field, Graph, GroupBox, Line, Oval, Picture, Rectangle, Report, RoundRectangle, TableBlob, and Text controls

Syntax

PocketBuilder dot notation:

dw_control.Object.*controlname*.Visible

Describe and Modify argument:

"*controlname*.Visible { = ' *value* ' }"

Parameter	Description
controlname	The name of the control for which you want to get or set the Visible property.
value	(*exp*) Whether the specified control is visible. Values are: 0 — False; the control is not visible. 1 — True; the control is visible. *Value* can be a quoted DataWindow expression.

Usage

In the painter Select the control and set the value in the Properties view, General tab.

Examples

```
string setting
setting = dw_1.Object.emp_status.Visible

dw_1.Object.emp_status.Visible = 0

dw_1.Object.emp_stat.Visible="0~tIf(emp_class=1,0,1)"

setting = dw_1.Describe("emp_status.Visible")

dw_1.Modify("emp_status.Visible=0")

dw_1.Modify("emp_stat.Visible='0~tIf(emp_class=1,0,1)'")
```

VTextAlign

Description

The way text in a button is vertically aligned.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Button controls

Syntax

PocketBuilder dot notation:

dw_control.Object.*buttonname*.VTextAlign

Describe and Modify argument:

"*buttonname*.VTextAlign { = ' *value* ' }"

Parameter	Description
buttonname	The name of the button for which you want to align text.
value	An integer indicating how the button text is horizontally aligned. Values are: 0 — Center 1 — Top 2 — Bottom 3 — Multiline

Usage

In the painter Select the control and set the value in the Properties view, General tab, Vertical Alignment option.

Examples

```
string setting
dw_1.Object.b_name.VTextAlign = "0"

setting = dw_1.Describe("b_name.VTextAlign")

dw_1.Modify("b_name.VTextAlign ='0'")
```

Width

Description

The width of the specified control.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Button, Column, Computed Field, Graph, GroupBox, Oval, Picture, Rectangle, Report, RoundRectangle, TableBlob, and Text controls

Syntax

PocketBuilder dot notation:

dw_control.Object.*controlname*.Width

Describe and Modify argument:

"*controlname*.Width { = ' *value* ' }"

Parameter	Description
controlname	The name of the control for which you want to get or set the width.

Parameter	Description
value	(*exp*) The width of the *controlname* in the units specified for the DataWindow. *Value* can be a quoted DataWindow expression.

Usage

In the painter Select the control and set the value in the Properties view, Position tab.

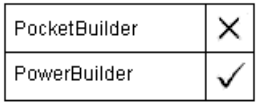
Examples

```
string setting
setting = dw_1.Object.emp_name.Width

dw_1.Object.emp_name.Width = 250

setting = dw_1.Describe("emp_name.Width")

dw_1.Modify("emp_name.Width=250")
```

Width.Autosize

Description

(RichText presentation style only) Whether the column or computed field input field adjusts its width according to the data it contains.



The Width.Autosize and Multiline properties can be set together so that the input field can display multiple lines.

Applies to

Column and Computed Field controls in the RichText presentation style

Syntax

PowerBuilder dot notation:

dw_control.Object.*controlname*.Width.Autosize

Describe and Modify argument:

"*controlname*.Width.Autosize { = ' *value* ' }"

X

Description

The distance of the specified control from the left edge of the DataWindow object.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Button, Column, Computed Field, Graph, GroupBox, Oval, Picture, Rectangle, Report, RoundRectangle, TableBlob, and Text controls

Syntax

PocketBuilder dot notation:

dw_control.Object.*controlname*.X

Describe and Modify argument:

"*controlname*.X { = ' *value* ' }"

Parameter	Description
controlname	The name of the control for which you want to get or set the x coordinate.
value	(*exp*) An integer specifying the x coordinate of the control in the unit of measure specified for the DataWindow object. *Value* can be a quoted DataWindow expression.

Usage

In the painter Select the control and set the value in the Properties view, Position tab.

Examples

```
string setting
setting = dw_1.Object.emp_name.X

dw_1.Object.emp_name.X = 10

setting = dw_1.Describe("emp_name.X")

dw_1.Modify("emp_name.X=10")
```

X1, X2

Description

The distance of each end of the specified line from the left edge of the line's band.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Line controls

Syntax

PocketBuilder dot notation:

dw_control.Object.*controlname*.X1

dw_control.Object.*controlname*.X2

Describe and Modify argument:

"*controlname*.X1 { = ' *value* ' }"

"*controlname*.X2 { = ' *value* ' }"

Parameter	Description
controlname	The name of the line for which you want to get or set one of the x coordinates.
value	(*exp*) An integer specifying the x coordinate of the line in the unit of measure specified for the DataWindow object. *Value* can be a quoted DataWindow expression.

Usage

In the painter Select the control and set the value in the Properties view, Position tab.

Examples

```
string setting
setting = dw_1.Object.line_1.X1

dw_1.Object.line_1.X1 = 10
dw_1.Object.line_1.X2 = 1000

setting = dw_1.Describe("line_1.X1")

dw_1.Modify("line_1.X1=10")
dw_1.Modify("line_1.X2=1000")
```

Y

Description

The distance of the specified control from the top of the control's band.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Button, Column, Computed Field, Graph, GroupBox, Oval, Picture, Rectangle, Report, RoundRectangle, TableBlob, and Text controls

Syntax

PocketBuilder dot notation:

dw_control.Object.*controlname*.Y

Describe and Modify argument:

"*controlname*.Y { = ' *value* ' }"

Parameter	Description
controlname	The name of the control for which you want to get or set the y coordinate.
value	(*exp*) An integer specifying the y coordinate of the control in the unit of measure specified for the DataWindow object. *Value* can be a quoted DataWindow expression.

Usage

In the painter Select the control and set the value in the Properties view, Position tab.

Examples

```
string setting
setting = dw_1.Object.emp_name.Y

dw_1.Object.emp_name.Y = 100

setting = dw_1.Describe("emp_name.Y")

dw_1.Modify("emp_name.Y=100")
```

Y1, Y2

Description

The distance of each end of the specified line from the top of the line's band.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Applies to

Line controls

Syntax

PocketBuilder dot notation:

dw_control.Object.*controlname*.Y1

dw_control.Object.*controlname*.Y2

Describe and Modify argument:

"*controlname*.Y1 { = ' *value* ' }"

"*controlname*.Y2 { = ' *value* ' }"

Parameter	Description
controlname	The name of the line for which you want to get or set one of the y coordinates.
value	(*exp*) An integer specifying the y coordinate of the line in the unit of measure specified for the DataWindow object. *Value* can be a quoted DataWindow expression.

Usage

In the painter Select the control and set the value in the Properties view, Position tab.

Examples

```
string setting
setting = dw_1.Object.line_1.Y1

dw_1.Object.line_1.Y1 = 50
dw_1.Object.line_1.Y2 = 50

setting = dw_1.Describe("line_1.Y1")

dw_1.Modify("line_1.Y1=50")
dw_1.Modify("line_1.Y2=50")
```

Zoom

Description

The scaling percentage of the DataWindow object.

PocketBuilder	✗
PowerBuilder	✓

Applies to

DataWindows

Syntax

PowerBuilder dot notation:

dw_control.Object.DataWindow.Zoom

Describe and Modify argument:

"DataWindow.Zoom { = *value* }"

CHAPTER 4

Accessing Data in Code

About this chapter

This chapter explains the syntax for constructing expressions that access data in a DataWindow object.

Contents

Accessing data and properties in DataWindow programming environments

In each programming environment, you can use methods and sometimes expressions to access the data and properties of a DataWindow object.

Data

Methods for single items of data These include GetItemString for data and Describe and Modify for properties. These methods are available in all environments.

DataWindow data expressions These let you access single items and blocks of data. You can access data in a single column, data in selected rows, and ranges of rows and columns.

Data expressions have a variety of syntaxes depending on the amount of data you want to access. Data expressions are not supported by the DataWindow Web Control for ActiveX.

You can get and set data values using the following syntax:

dwcontrol.**Object.Data** [*startrownum, startcolnum, endrownum, endcolnum*]

For a list of syntaxes, see “Syntaxes for DataWindow data expressions” on page 332.

Properties

Methods for properties These are Describe and Modify. These methods are available in all environments.

DataWindow property expressions These let you get and set the values of properties of the DataWindow definition and of controls contained within the definition, such as columns and text labels. Property expressions are not supported by the DataWindow Web Control for ActiveX.

Property expressions take this form:

dwcontrol.**Object**.*columnname*.*columnproperty* = value

Where to find information

This chapter discusses techniques for accessing data with emphasis on data expressions.

For information on accessing properties using methods or property expressions, see Chapter 5, "Accessing DataWindow Object Properties in Code."

Techniques for accessing data

Two techniques

There are two ways to access data values in a DataWindow control:

- **Methods** SetItem and the group of GetItem methods access single values in specific rows and columns. For example:

```
dw_1.SetItem(1, "empname", "Phillips")
ls_name = dw_1.GetItemString(1, "empname")
```

- **Expressions** DataWindow data expressions use dot notation and can refer to single items, columns, blocks of data, selected data, or the whole DataWindow control. For example:

```
dw_1.Object.empname[1] = "Phillips"
dw_1.Object.Data[1,1] = "Phillips"
```

Both methods allow you to access data in any buffer and to get original or current values.

Which technique to use

The technique you use depends on how much data you are accessing and whether you know the names of the DataWindow columns when the script is compiled:

Table 4-1: Which technique to use when accessing data

If you want to access	Use
A single item	Either an expression or a method. Both are equally efficient when referring to single items. **Exception** If you want to use a column's name rather than its number, and the name is not known until runtime, use a method; methods allow you to name the column dynamically.
More than one item, such as: • All the data in a column • A block of data specified by ranges of rows and columns • Data in selected rows • All the data in the DataWindow	An expression. Specifying the data you want in a single statement is much more efficient than calling the methods repeatedly in a program loop.

What's in this section

The rest of this section describes how to construct expressions for accessing DataWindow data. The section "Syntaxes for DataWindow data expressions" on page 332 provides reference information on the syntaxes for data expressions.

For information on methods

For information about using methods for accessing data, see SetItem, GetItemDate, GetItemDateTime, GetItemDecimal, GetItemNumber, GetItemString, and GetItemTime in Chapter 9, "Methods for the DataWindow Control."

About DataWindow data expressions

The Object property of the DataWindow control lets you specify expressions that refer directly to the data of the DataWindow object in the control. This direct data manipulation allows you to access small and large amounts of data in a single statement, without calling methods.

There are several variations of data expression syntax, divided into three groups. This section summarizes these syntaxes. The syntaxes are described in detail later in this chapter.

Data in columns or computed fields when you know the name

One or all items (if rownum is absent, include either *buffer* or *datasource*)

dwcontrol.Object.*columnname* {.*buffer* } {.*datasource* } { [*rownum*] }

Returns a single value (for a specific row number) or an array of values (when *rownum* is omitted) from the column.

See "Syntax for one or all data items in a named column" on page 333.

Selected items

dwcontrol.Object.*columnname* {.Primary }{.*datasource* }.Selected

Returns an array of values from the column with an array element for each selected row.

See "Syntax for selected data in a named column" on page 335.

Range of items

dwcontrol.Object.*columnname* {.*buffer* } {.*datasource* } [*startrownum*, *endrownum*]

Returns an array of values from the column with an array element for each row in the range.

See "Syntax for a range of data in a named column" on page 337.

Data in numbered columns

Single items

dwcontrol.Object.Data {.*buffer* } {.*datasource* } [*rownum*, *colnum*]

Returns a single item whose datatype is the datatype of the column.

See "Syntax for a single data item in a DataWindow" on page 339.

Blocks of data involving a range of rows and columns

dwcontrol.Object.Data {.*buffer* } {.*datasource* } [*startrownum*, *startcolnum*, *endrownum*, *endcolnum*]

Returns an array of structures or user objects. The structure elements match the columns in the range. There is one array element for each row in the range.

See "Syntax for data in a block of rows and columns" on page 340.

Whole rows

Single row or all rows

dwcontrol.Object.Data {.*buffer* } {.*datasource* } { [*rownum*] }

Returns one structure or user object (for a single row) or an array of them (for all rows). The structure elements match the columns in the DataWindow object.

See "Syntax for data in a single row or all rows" on page 342.

Selected rows

dwcontrol.Object.Data {.Primary } {.*datasource* } .Selected

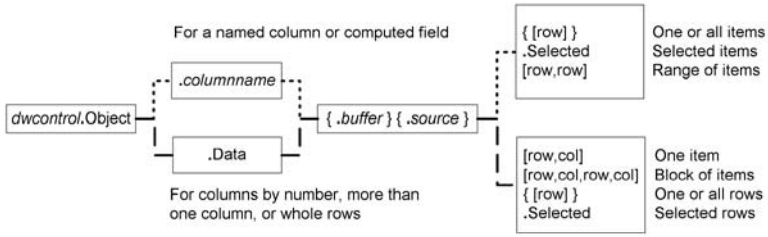
Returns an array of structures or user objects. The structure elements match the columns in the DataWindow object. There is one array element for each selected row.

See “Syntax for all data from selected rows” on page 344.

Summary of syntaxes

This diagram summarizes the variations in data expression syntax:

Figure 4-1: Variations in data expression syntax



For information about getting and setting values of DataWindow object properties using a similar syntax, see Chapter 5, “Accessing DataWindow Object Properties in Code.”

When a DataWindow data expression is evaluated

Expressions that refer to DataWindow data are not verified until execution of your application.

No compiler checking

When your script is compiled, PocketBuilder does not verify the parameters of the expression that follow the Object property. Your application can select or change the DataWindow object in a DataWindow control during execution without invalidating the compiled script.

Potential execution errors

If the datatype of the expression is not compatible with how the expression is used, or if the specified rows or columns do not exist, an error will occur during execution.

You can handle the error by surrounding the expression in a try-catch block and catching any DWRuntimeErrors, or by writing a script for the DataWindow control’s Error event.

Getting and storing the data from a DataWindow data expression

A DataWindow data expression can return a large amount of data.

Data structures for data

Single row and column When your data expression refers to a single row and column, you can assign the data to a variable whose data matches the column's datatype. When the expression refers to a single column but can refer to multiple rows, you must specify an array of the appropriate datatype.

More than one column When the expression refers to more than one column, you can get or set the data with a structure or user object. When you create the definition, you must assign datatypes to the fields (in a structure) or instance variables (in a user object) that match the datatypes of the columns. When your expression refers to multiple rows, you get an array of the structure or user object.

Likewise, if you want to set data in the DataWindow control, you will set up the data in structures or user objects whose elements match the columns referred to in the expression. An array of those structures or user objects will provide data for multiple rows.

Datatypes For matching purposes, the datatypes should be appropriate to the data—for example, any numeric datatype matches any other numeric type.

Examples of data structures

The following table presents some examples of data specified by an expression and the type of data structures you might define for storing the data:

Table 4-2: Types of storage for data specified by an expression

Type of selection	Sample data storage
A single item	A single variable of the appropriate datatype.
A column of values	An array of the appropriate datatype.
A row	A structure whose elements have datatypes that match the DataWindow object's columns. A user object whose instance variables match the DataWindow object's columns.
Selected rows or all rows	An array of the structure or user object defined for a row.
A block of values	An array of structures or user objects whose elements or instance variables match the columns included in the selected range.

Assigning data to arrays

When a data expression is assigned to an array, values are assigned beginning with array element 1 regardless of the starting row number. If the array is larger than the number of rows accessed, elements beyond that number are unchanged. If it is smaller, a variable-size array will grow to hold the new values. However, a fixed-size array that is too small for the number of rows will cause an execution error.

Two ways to instantiate user objects
A user object needs to be instantiated before it is used.

One way is to use the CREATE statement after you declare the user object. If you declare an array of the user object, you must use CREATE for each array element.

The second way is to select the Autoinstantiate box for the user object in the User Object painter. When you declare the user object in a script, the user object will be automatically instantiated, like a structure.

Any datatype and data expressions

The actual datatype of a DataWindow data expression is Any, which allows the compiler to process the expression even though the final datatype is unknown. When data is accessed at runtime, you can assign the result to another Any variable or to a variable, structure, or user object whose datatype matches the real data.

Examples

A single value This example gets a value from column 2, whose datatype is string:

```
string ls_name
ls_name = dw_1.Object.Data[1,2]
```

A structure that matches DataWindow columns In this example, a DataWindow object has four columns:

An ID (number)
A name (string)
A retired status (boolean)
A birth date (date)

A structure to hold these values has been defined in the Structure painter. It is named str_empdata and has four elements whose datatypes are integer, string, boolean, and date. To store the values of an expression that accesses some or all the rows, you need an array of str_empdata structures to hold the data:

```
str_empdata lstr_currdata[]
lstr_currdata = dw_1.Object.Data
```

After this example executes, the upper bound of the array of structures, which is variable-size, is equal to the number of rows in the DataWindow control.

A user object that matches DataWindow columns If the preceding example involved a user object instead of a structure, then a user object defined in the User Object painter, called uo_empdata, would have four instance variables, defined in the same order as the DataWindow columns:

```
integer id
string name
boolean retired
date birthdate
```

Before accessing three rows, three array elements of the user object have been created (you could use a FOR NEXT loop for this). The user object was not defined with Autoinstantiate enabled:

```
uo_empdata luo_empdata[3]
luo_empdata[1] = CREATE uo_empdata
luo_empdata[2] = CREATE uo_empdata
luo_empdata[3] = CREATE uo_empdata
luo_empdata = dw_1.Object.Data[1,1,3,4]
```

Setting DataWindow data with a DataWindow data expression

When you set data in a DataWindow control, the datatypes of the source values must match the datatypes of the columns being set.

Single value or an array

When your data expression refers to a single row and column, you can set the value in the DataWindow control with a value that matches the column's datatype. When you are setting values in a single column and specifying an expression that can refer to multiple rows, the values you assign must be in an array of the appropriate datatype.

Multiple columns and whole rows

When the expression refers to more than one column, you can assign the data with a structure or user object to the DataWindow data. When you create the definition, the fields (in a structure) or instance variables (in a user object) must match the columns. There must be the same number of fields or variables, defined in the same order as the columns, with compatible datatypes.

When your expression can refer to multiple rows, you need an array of the structure or user object.

Using arrays to set values

You do not have to know in advance how many rows are involved when you are setting data in the DataWindow control. PocketBuilder uses the number of elements in the source array and the number of rows in the target expression to determine how to make the assignment and whether it is necessary to insert rows.

If the target expression is *selected rows or a range of rows*, then:

- When there are *more* array elements than target rows, the extra array elements are ignored
- When there are *fewer* array elements than target rows, the column(s) in the extra target rows are filled with default values

If the target expression is *all rows but not all columns*, then:

- When there are *more* array elements than target rows, the extra array elements are ignored
- When there are *fewer* array elements than target rows, only the first rows up to the number of array elements are affected

If the target expression is *all rows and all columns*, then the source data replaces all the existing rows, resetting the DataWindow control to the new data.

Inserting new rows When you are setting data and you specify a range, then if rows do not exist in that range, rows are inserted to fill the range. For example, if the DataWindow control has four rows and your expression says to assign data to rows 8 through 12, then eight more rows are added to the DataWindow control. The new rows use the initial default values set up for each column. After the rows are inserted, the array of source data is applied to the rows as described above.

Examples

These examples refer to a DataWindow object that has three columns: emp_id, emp_lname, and salary. The window declares these arrays as instance variables and the window's Open event assigns four elements to each array:

```
integer ii_id[]
string is_name[]
double id_salary[]
uo_empdata iuo_data[]
uo_empid_name iuo_id[]
```

The uo_empdata user object has three instance variables: id, name, and salary. The uo_empid_name user object has two instance variables: ID and name.

This example sets emp_lname in the selected rows to the values of is_name, an array with four elements. If two rows are selected, only the first two values of the array are used. If six rows are selected, the last two rows of the selection are set to an empty string:

```
dw_1.Object.emp_lname.Selected = is_name
```

This example sets salary in rows 8 to 12 to the values in the array id_salary. The id_salary array has only four elements, so the extra row in the range is set to 0 or a default value:

```
dw_1.Object.salary[8,12] = id_salary
```

This statement resets the DataWindow control and inserts four rows to match the array elements of iuo_data:

```
dw_1.Object.Data.Primary = iuo_data
```

This example sets columns 1 and 2 in rows 5 to 8 to the values in the array iuo_id:

```
dw_1.Object.Data.Primary[5,1, 8,2] = iuo_id
```

This example sets emp_id in the first four rows to the values in the ii_id array. Rows 5 through 12 are not affected:

```
dw_1.Object.emp_id.Primary = ii_id
```

Syntaxes for DataWindow data expressions

This section describes in detail the syntaxes that were summarized in “About DataWindow data expressions” on page 325.

You can think of the syntaxes as grouped in three categories:

- Expressions with a named column or computed field
  - “Syntax for one or all data items in a named column” on page 333
  - “Syntax for selected data in a named column” on page 335
  - “Syntax for a range of data in a named column” on page 337
- Expressions with column numbers
  - “Syntax for a single data item in a DataWindow” on page 339
  - “Syntax for data in a block of rows and columns” on page 340
- Expressions that access whole rows
  - “Syntax for data in a single row or all rows” on page 342
  - “Syntax for all data from selected rows” on page 344

Syntax for one or all data items in a named column

Description

A DataWindow data expression can access a single item in a column or computed field when you specify the control name and a row number. It accesses all the data in the column when you omit the row number.

Syntax

dwcontrol.Object.*columnname* {.*buffer* } {.*datasource* } { [*rownum*] }

Parameter	Description
dwcontrol	The name of the DataWindow control or child DataWindow in which you want to get or set data.
columnname	The name of a column or computed field in the DataWindow object in *dwcontrol*. If the column or computed field does not exist at runtime, an execution error occurs.
buffer (optional)	The name of the buffer from which you want to get or set data. Values are: • Primary — (Default) The data in the primary buffer (the data that has not been deleted or filtered out). • Delete — The data in the delete buffer (data deleted from the DataWindow control). • Filter — The data in the filter buffer (data that was filtered out).
datasource (optional)	The source of the data. Values are: • Current — (Default) The current values in the DataWindow control. • Original — The values that were initially retrieved from the database. For a computed field, you must specify Original because computed fields cannot be changed and do not have current values.
rownum (optional)	The row number of the desired item. The row number must be enclosed in brackets. To access all the data in the column, omit *rownum*. **When buffer or datasource is not optional** When *rownum* is omitted, you must specify at least one of the other elements in the expression: either *buffer* or *datasource*.

Return value

The expression has a datatype of Any. The expression returns a single value (for a specific row number) or an array of values (when *rownum* is omitted). Each value has a datatype of *columnname*.

Usage

Is the expression a DWObject or data? When you want to access all the data in the column, remember to specify at least one of the other optional parameters. Otherwise, the expression you specify refers to the column *control*, not its data. This expression refers to the DWObject empname, not the data in the column:

```
dw_1.Object.empname
```

In contrast, these expressions all refer to data in the empname column:

```
dw_1.Object.empname.Primary // All rows
dw_1.Object.empname[5]      // Row 5
```

Row numbers for computed fields When you refer to a control in a band other than the detail band (usually a computed field) you still specify a row number. For the header, footer, or summary, specify a row number of 1. For the group header or trailer, specify the group number:

```
dw_1.Object.avg_cf[1]
```

If you specify nothing after the computed field name, you refer to the computed field DWObject, not the data. For a computed field that occurs more than once, you can get all values by specifying *buffer* or *datasource* instead of *rownum*, just as for columns.

When the expression is an array When the expression returns an array (because there is no row number), you must assign the result to an array, even if you know there is only one row in the result.

This expression returns an array, even if there is only one row in the DataWindow control:

```
dw_1.Object.empname.Primary
```

This expression returns a single value:

```
dw_1.Object.empname[22]
```

Examples

Because the default setting is current values in the primary buffer, the following expressions are equivalent—both get the value in row 1 for the emp_name column:

```
dw_1.Object.emp_name[1]
dw_1.Object.emp_name.Primary.Current[1]
```

This statement sets the emp_name value in row 1 to Wilson:

```
dw_1.Object.emp_name[1] = "Wilson"
```

This statement gets values for all the emp_name values that have been retrieved and assigns them to an array of strings:

```
string ls_namearray[]
ls_namearray = dw_1.Object.emp_name.Current
```

This statement gets current values of emp_name from all rows in the filter buffer:

```
string ls_namearray[]
ls_namearray = dw_1.Object.emp_name.Filter
```

This statement gets original values of emp_name from all rows in the filter buffer:

```
string ls_namearray[]
ls_namearray = dw_1.Object.emp_name.Filter.Original
```

This statement gets the current value of emp_name from row 14 in the delete buffer:

```
string ls_name
ls_name = dw_1.Object.emp_name.Delete[14]
```

This statement gets the original value of emp_name from row 14 in the delete buffer:

```
string ls_name
ls_name = dw_1.Object.emp_name.Delete.Original[14]
```

This statement gets all the values of the computed field review_date:

```
string ld_review[]
ld_review = dw_1.Object.review_date.Original
```

Syntax for selected data in a named column

Description

A DataWindow data expression uses the Selected property to access values in a named column or computed field for the currently selected rows. Selected data is always in the primary buffer.

Syntax

dwcontrol.Object.*columnname* {.Primary } {.*datasource* }.Selected

Parameter	Description
dwcontrol	The name of the DataWindow control or child DataWindow in which you want to get or set data.
columnname	The name of a column or computed field in the DataWindow object in *dwcontrol*. If the column or computed field does not exist at runtime, an execution error occurs.
datasource (optional)	The source of the data. Values are: • Current — (Default) The current values in the DataWindow control. • Original — The values that were initially retrieved from the database. For a computed field, you must specify Original (because computed fields cannot be changed and do not have current values).

Return value

The datatype of the expression is Any. The expression returns an array of values with the datatype of *columnname*.

Usage

When you specify selected values, the expression always returns an array and you must assign the result to an array, even if you know there is only one row selected.

For selected rows, the primary buffer is the only applicable buffer. For consistency, you can include Primary in this syntax, but it is not necessary.

Examples

Because the primary buffer is the only applicable buffer for selected data and current data is the default, these expressions are all equivalent. They access values in the emp_name column for selected rows:

```
dw_1.Object.emp_name.Selected
dw_1.Object.emp_name.Primary.Selected
dw_1.Object.emp_name.Current.Selected
dw_1.Object.emp_name.Primary.Current.Selected
```

These expressions both access original values for selected rows:

```
dw_1.Object.emp_name.Original.Selected
dw_1.Object.emp_name.Primary.Original.Selected
```

This example sets the emp_name value in the first selected row to an empty string. The rest of the selected rows are set to a default value, which can be an empty string:

```
string ls_empty[]
ls_empty[1] = ""
dw_1.Object.emp_lname.Selected = ls_empty
```

This statement gets the original emp_name values in selected rows and assigns them to an array of strings:

```
string ls_namearray[]
ls_namearray = dw_1.Object.emp_name.Original.Selected
```

Syntax for a range of data in a named column

Description

A DataWindow data expression accesses values in a named column or computed field for a range of rows when you specify the starting and ending row numbers.

Syntax

dwcontrol.Object.*columnname* {.*buffer* } {.*datasource* } [*startrownum*, *endrownum*]

Parameter	Description
dwcontrol	The name of the DataWindow control or child DataWindow in which you want to get or set data.
columnname	The name of a column or computed field in the DataWindow object in *dwcontrol*. If the column or computed field does not exist at runtime, an execution error occurs.
buffer (optional)	The name of the buffer from which you want to get or set data. Values are: • Primary — (Default) The data in the primary buffer (the data that has not been deleted or filtered out). • Delete — The data in the delete buffer (data deleted from the DataWindow control). • Filter — The data in the filter buffer (data that was filtered out).
datasource (optional)	The source of the data. Values are: • Current — (Default) The current values in the DataWindow control. • Original — The values that were initially retrieved from the database. For a computed field, you must specify Original (because computed fields cannot be changed and do not have current values).
startrownum	The number of the first row in the desired range of rows.
endrownum	The number of the last row in the desired range of rows. The row numbers must be enclosed in brackets and separated by commas.

Return value

The datatype of the expression is Any. The expression returns an array of values with an array element for each row in the range. Each value's datatype is the datatype of *columnname*.

Usage

When you specify a range, the expression always returns an array and you must assign the result to an array, even if you know there is only one value in the result. For example, this expression returns an array of one value:

```
dw_1.Object.empname[22,22]
```

Examples

Because the primary buffer and current data are the default, these expressions are all equivalent:

```
dw_1.Object.emp_name[11,20]
dw_1.Object.emp_name.Primary[11,20]
dw_1.Object.emp_name.Current[11,20]
dw_1.Object.emp_name.Primary.Current[11,20]
```

This example resets the emp_name value in rows 11 through 20 to an empty string. Rows 12 to 20 are set to a default value, which may be an empty string:

```
string ls_empty[]
ls_empty[1] = ""
dw_1.Object.emp_name[11,20] = &
       {"","","","","","","","","",""}
```

This statement gets the original emp_name values in rows 11 to 20 and assigns them to elements 1 to 10 in an array of strings:

```
string ls_namearray[]
ls_namearray = dw_1.Object.emp_name.Original[11,20]
```

This statement gets current values of emp_name from rows 5 to 8 in the Filter buffer and assigns them to elements 1 to 4 in an array of strings:

```
string ls_namearray[]
ls_namearray = dw_1.Object.emp_name.Filter[5,8]
```

This statement gets original values of emp_name instead of current values, as shown in the previous example:

```
string ls_namearray[]
ls_namearray = &
       dw_1.Object.emp_name.Filter.Original[5,8]
```

This statement gets current values of emp_name from rows 50 to 200 in the delete buffer and assigns them to elements 1 to 151 in an array of strings:

```
string ls_namearray[]
ls_namearray = dw_1.Object.emp_name.Delete[50,200]
```

This statement gets original values of emp_name instead of current values, as shown in the previous example:

```
string ls_namearray[]
ls_namearray = &
       dw_1.Object.emp_name.Delete.Original[50,200]
```

Syntax for a single data item in a DataWindow

Description

A DataWindow data expression accesses a single data item when you specify its row and column number.

Syntax

dwcontrol.Object.Data {.*buffer* } {.*datasource* } [*rownum*, *colnum*]

Parameter	Description
dwcontrol	The name of the DataWindow control or child DataWindow in which you want to get or set data.
buffer (optional)	The name of the buffer from which you want to get or set data. Values are: • Primary — (Default) The data in the primary buffer (the data that has not been deleted or filtered out). • Delete — The data in the delete buffer (data deleted from the DataWindow control). • Filter — The data in the filter buffer (data that was filtered out).
datasource (optional)	The source of the data. Values are: • Current — (Default) The current values in the DataWindow control. • Original — The values that were initially retrieved from the database.
rownum	The row number of the desired item.
colnum	The column number of the desired item. The row and column numbers must be enclosed in brackets and separated by commas.

Return value

The datatype of the expression is Any. The expression returns a single item in the DataWindow control. Its datatype is the datatype of the column.

Examples

These expressions both refer to a single item in row 1, column 2. The expressions access current data in the primary buffer:

```
dw_1.Object.Data[1,2]
dw_1.Object.Data.Primary.Current[1,2]
```

This statement changes the value of the original data to 0 for the item in row 1, column 2 in the Filter buffer. Column 2 holds numeric data:

```
dw_1.Object.Data.Filter.Original[1,2] = 0
```

Syntax for data in a block of rows and columns

Description

A DataWindow data expression accesses data in a range of rows and columns when you specify the starting and ending row and column numbers.

Syntax

dwcontrol.Object.Data {.*buffer* } {.*datasource* } [*startrownum*, *startcolnum*, *endrownum*, *endcolnum*]

Parameter	Description
dwcontrol	The name of the DataWindow control or child DataWindow in which you want to get or set data.
buffer (optional)	The name of the buffer from which you want to get or set data. Values are: • Primary — (Default) The data in the primary buffer (the data that has not been deleted or filtered out). • Delete — The data in the delete buffer (data deleted from the DataWindow control). • Filter — The data in the filter buffer (data that was filtered out).
datasource (optional)	The source of the data. Values are: • Current — (Default) The current values in the DataWindow control. • Original — The values that were initially retrieved from the database.
startrownum	The number of the first row in the desired range of rows.
startcolnum	The number for the first column in the range.
endrownum	The number of the last row in the range.
endcolnum	The number for the last column in the range. The row and column numbers must be enclosed in brackets and separated by commas.

Return value

The datatype of the expression is Any. The expression returns an array of structures or user objects. There is one structure element or user object instance variable for each column in the designated range. The datatype of each element matches the datatype of the corresponding column. There is one structure or user object in the array for each row in the range of rows.

Usage

When you specify a block, the expression always returns an array and you must assign the result to an array, even if you know there is only one structure in the result.

This expression returns an array of one structure from row 22:

```
dw_1.Object.data[22,1,22,4]
```

This expression returns an array of one value from row 22, column 1:

```
dw_1.Object.data[22,1,22,1]
```

Examples

These statements both refer to data in the first ten rows and first four columns of the DataWindow object in the control dw_1. The primary buffer and current data are the default:

```
dw_1.Object.Data[1,1,10,4]
dw_1.Object.Data.Primary.Current[1,1,10,4]
```

This example gets employee IDs and last names for all the rows in the delete buffer. The IDs and names are the first two columns. It saves the information in a structure, called str_namelist, of two elements: an integer called id and a string called lastname. The structure was defined previously in the Structure painter. The list of IDs and names is then saved in the file DELETED.TXT:

```
integer li_fileNum
long ll_deletedrows
str_namelist lstr_namelist[]

ll_deletedrows = dw_1.DeletedCount()
lstr_namelist = &
       dw_1.Object.Data.Delete[1,1, ll_deletedrows,2]

li_fileNum = FileOpen("C:\HR\DELETED.TXT", &
       LineMode!, Write!)
FOR ll_count = 1 to UpperBound(lstr_namelist)
       FileWrite(li_fileNum, &
           String(lstr_namelist.id) + &
           " " + &
           lstr_namelist.lastname + &
           "~r~n")
NEXT
FileClose(li_fileNum)
```

Using the structure from the previous example that holds IDs and last names, this example sets all the IDs and last names in the DataWindow control to NULL:

```
long ll_n
str_namelist lstr_namelist[]

SetNull(lstr_namelist[1].id)
SetNull(lstr_namelist[1].lastname)

FOR ll_n = 2 to dw_1.RowCount()
       lstr_namelist[ll_n] = lstr_namelist[1]
NEXT

dw_1.Object.Data[1,1, dw_1.RowCount(),2] = lstr_data
```

Syntax for data in a single row or all rows

Description

A DataWindow data expression accesses a single row when you specify the row number. It accesses all the data in the DataWindow control when you omit the row number.

Syntax

dwcontrol.Object.Data {.*buffer* } {.*datasource* } { [*rownum*] }

Parameter	Description
dwcontrol	The name of the DataWindow control or child DataWindow in which you want to get or set data.
buffer (optional)	The name of the buffer from which you want to get or set data. Values are: • Primary — (Default) The data in the primary buffer (the data that has not been deleted or filtered out). • Delete — The data in the delete buffer (data deleted from the DataWindow control). • Filter — The data in the filter buffer (data that was filtered out).

Parameter	Description
datasource (optional)	The source of the data. Values are: • Current — (Default) The current values in the DataWindow control. • Original — The values that were initially retrieved from the database.
rownum (optional)	The number of the row you want to access. To access data for all rows, omit *rownum*. The row number must be enclosed in brackets.

Return value

The datatype of the expression is Any. The expression returns one structure or user object (for a single row) or an array of them (for all rows). There is one structure element or instance variable for each column in the DataWindow object. The datatype of each element matches the datatype of the corresponding column.

Usage

When you omit the row number, the expression always returns an array, and you must assign the result to an array even if you know there is only one row in the DataWindow control.

Examples

These statements both access current data for row 5 in the primary buffer in the DataWindow object contained in the DataWindow control dw_1:

```
dw_1.Object.Data[5]
dw_1.Object.Data.Primary.Current[5]
```

This example assigns all the data in dw_1 to the Any variable la_dwdata. The value assigned to la_dwdata is an array of data structures whose members match the column datatypes:

```
any la_dwdata
la_dwdata = dw_1.Object.Data
```

This example assigns all the data in the delete buffer for dw_1 to the Any variable la_dwdata:

```
any la_dwdata
la_dwdata = dw_1.Object.Data.Delete
```

This example replaces all the data in the nested report in row 2 with data from dw_2. The columns in the DataWindow object in dw_2 must match the columns in the DataWindow object for the nested report:

```
dw_1.Object.NestRep[2].Object.Data = &
       dw_2.Object.Data
```

Syntax for all data from selected rows

Description

A DataWindow data expression accesses all the data in the currently selected rows when you specify the Data and Selected properties. Selected rows are always in the primary buffer.

Syntax

dwcontrol.Object.Data {.Primary } {.*datasource* }.Selected

Parameter	Description
dwcontrol	The name of the DataWindow control or child DataWindow in which you want to get or set data.
datasource (optional)	The source of the data. Values are: • Current — (Default) The current values in the DataWindow control. • Original — The values that were initially retrieved from the database.

Return values

The datatype of the expression is Any. The expression returns an array of structures or user objects. There is one structure element or instance variable for each column in the DataWindow object. The datatype of each element matches the datatype of the corresponding column.

Usage

When you specify selected rows, the expression always returns an array, and you must assign the result to an array even if you know there is only one row selected.

Examples

Because the primary buffer is the only applicable buffer for selected data and current data is the default, these expressions are all equivalent. They access data in the selected rows:

```
dw_1.Object.Data.Selected
dw_1.Object.Data.Primary.Selected
dw_1.Object.Data.Current.Selected
dw_1.Object.Data.Primary.Current.Selected
```

Both these expressions access original values for selected rows:

```
dw_1.Object.Data.Original.Selected
dw_1.Object.Data.Primary.Original.Selected
```

This example takes the values in the selected rows in dw_2 and populates a DropDownDataWindow in dw_1 with the values, replacing existing data in the DropDownDataWindow. The column with the DropDownDataWindow is called useroptions. The columns of the DataWindow object in dw_2 must match the columns of the DataWindow object for the DropDownDataWindow:

```
dw_1.Object.useroptions.Object.Data = &
       dw_2.Object.Data.Selected
```

CHAPTER 5

Accessing DataWindow Object Properties in Code

About this chapter

This chapter explains the syntax for constructing expressions that access properties of controls within a DataWindow.

Contents

About properties of the DataWindow object and its controls

This section describes:

- What you can do with DataWindow object properties
- Specifying property values in the DataWindow painter
- Accessing DataWindow object property values in code
- Using DataWindow expressions as property values
- Nested strings and special characters for DataWindow object properties

What you can do with DataWindow object properties

The DataWindow object defines the way data is displayed in a DataWindow control. It contains controls that represent the columns, text labels, computed fields, and images.

The properties of the DataWindow object and its controls store the information that specifies the behavior of the DataWindow object. They are not properties of the DataWindow control, but of the DataWindow object displayed in the control.

Terminology

When you are programming for DataWindows, there are several types of expressions involved.

A **DataWindow expression** is an expression assigned as a value to a DataWindow property and is evaluated by the DataWindow engine. The expression can refer to column data and can have a different value for each row in the DataWindow.

A **DataWindow property expression** is an expression in your code that gets or sets the value of a DataWindow property. Its effects are equivalent to what the Describe and Modify methods do.

A **DataWindow data expression** is an expression in your code that gets or sets data in the DataWindow. Its effects are similar to what the SetItem and several GetItem methods do.

Types of values

Property values can be constants or DataWindow expressions. DataWindow expressions allow the property value to be based on other conditions in the DataWindow, including data values. Conditional expressions based on data can give the property a different value for each row.

Getting and setting values

You establish initial values for properties in the DataWindow painter. You can also get and set property values during execution in code.

There are several techniques for accessing property values. A particular property might be accessible by a subset of those techniques. For example, some properties are read-only during execution, some can be set only at execution, and some accept only constants (not DataWindow expressions) as values.

For a complete list of properties and the ways you can access each one, see Chapter 3, “DataWindow Object Properties.”

Examples: ways of setting the Border property

This table lists the ways you can access a property, using the Border property as an example:

Table 5-1: Ways to access and change DataWindow object properties

What you can do with properties	How to do it, using the Border property as an example	What happens
Set the initial value of the property in the workspace	Property sheet, General tab, Border box	The Border property takes on the value you set unconditionally. In the Preview view and at runtime, the control has the border you indicated in the workspace unless you set the Border property again in some way.
Specify the value of the property at runtime based on an expression defined for the control in the workspace	Property sheet, General tab, Border box, Expression button	In Preview and at runtime, the border changes as specified in the expression, which overrides the setting on the property sheet. For example, an expression can give the Salary column value a ShadowBox border when the salary exceeds $70,000. To see the effect in the Preview view, you might need to close Preview and reopen it.
Get the value of the property at runtime in code	Property expression for the Border property *or* Describe method	Both the expression and the Describe method return the value of the Border property for the specified control.
Change the value of the property at runtime in code	Property expression for the Border property *or* Modify method	At runtime, the value of the property changes when the code executes. For example, you could code Modify in the Clicked event and change the border of the control the user clicked.
Set the initial value of the property at runtime in code for a DataWindow being created	SyntaxFromSQL method	When SyntaxFromSQL executes, the border value of all columns is set in the generated syntax. SyntaxFromSQL is a method of the Transaction object and is described in the *PowerScript Reference*.

Specifying property values in the DataWindow painter

When you specify values in the Properties view of the DataWindow painter, you are setting properties of the DataWindow object and its controls.

Properties for each control

Each control in the DataWindow (columns, text, drawing controls) has its own property sheets, because there are different sets of properties for each object. To access individual property sheets, display the Properties view and then select a control.

If several controls have the same property and you want them all to have the same value, you can select all the controls so that the property sheet shows the properties they have in common. When you change the property value, it is applied to all selected controls.

DataWindow expressions for properties

For many properties, you can specify a DataWindow expression in the Properties view by clicking the Expression button beside the property. At runtime, the expression is evaluated for each row. When the expression includes row-dependent information in the calculation (such as data), each row can have a different value for the property. In the painter, you can see the results in the Preview view. (You might need to close Preview and reopen it if you are not seeing the settings you have made.)

For information about the components of expressions, see "Using DataWindow expression functions" on page 15 and the *Users Guide*. For examples of expressions, see "Using DataWindow expressions as property values" on page 349.

Accessing DataWindow object property values in code

Two techniques

There are two ways to access property values in a DataWindow object:

- **Methods** The Describe and Modify methods use strings to specify the property names. For example:

```
dw_1.Describe("empname.Border")
dw_1.Modify("empname.Border=1")
```

- **Expressions** DataWindow property expressions use the Object property and dot notation. For example:

```
dw_1.Object.empname.Border = 1
li_border = Integer(dw_1.Object.empname.Border)
```

Which technique to use

The technique you use depends on the type of error checking you want to provide and on whether you know the names of the controls and properties you want to access when the script is compiled.

Table 5-2: Error handling in DataWindow property expressions

If you want to	Use
Use column and property names that are known when the script is compiled	An expression
Avoid extra nested tildes (and you know the column and property names you want to access)	An expression
Build a string at runtime that names controls and properties	A method
Use the DWRuntimeError to handle problems with incorrect control or property names	An expression in a try-catch block
Use the Error event to handle problems with incorrect control or property names	An expression and a script for the Error event
Avoid using the Error event (or DWRuntimeError) for handling problems with incorrect control or property names	A method and code that evaluates its return value

Using DataWindow expressions as property values

When a DataWindow object property's value can be an expression, you can make the control's appearance or other properties depend on other information in the DataWindow.

A DataWindow expression can include:

- Operators.
- The names of controls within the DataWindow, especially column and computed field names.
- DataWindow expression functions. Some functions, such as IsRowNew, refer to characteristics of an individual row.
- User-defined functions.

Different formats for the expression

When you assign an expression in the painter, you specify just the expression:

DataWindowexpression

When you assign an expression in code, you specify a default value, a tab, and the expression:

defaultvalue [tab] *DataWindowexpression*

Examples

In the painter This expression for a column called emp_lname is applied to the Background.Color property. It causes the name's background to be light gray (15790320) if the current row (person) uses the day care benefit. If not, the background color is set to white:

```
If(bene_day_care = 'Y', 15790320, 1677215)
```

In code The expression assigned to the Background.Color property includes a default value. Nested quotes complicate the syntax:

```
dw_1.Object.emp_lname.Background.Color = "16777215 ~t
If(bene_day_care = 'Y', 15790320, 16777215)"
```

More examples in the DataWindow painter and in code

These examples illustrate the difference between the format for a DataWindow expression specified in the DataWindow painter versus in code.

Border property The expression applied to the Border property of the salary_plus_benefits column displays a border around salaries over $60,000:

```
If(salary_plus_benefits > 60000, 1, 0)
```

This statement changes the expression in code:

```
dw_1.Object.salary_plus_benefits.Border = &
       "0 ~t If(salary_plus_benefits > 60000, 1, 0)"
```

Font.Weight property for a column To make out-of-state (not in Massachusetts) names and numbers bold in a phone list, apply this expression to the name and phone_number columns. The state column must be part of the data source, but it does not have to be displayed:

```
If(state = 'MA', 400, 700)
```

This statement changes the expression in code:

```
dw_1.Object.name.Font.Weight = &
       "700 ~t If(state = 'MA', 400, 700)"
dw_1.Object.phone_number.Font.Weight = &
       "700 ~t If(state = 'MA', 400, 700)"
```

Brush.Color property for a rectangle This expression, applied to a rectangle drawn around all the columns in a tabular report, causes alternate rows to be shaded (a graybar effect). Make sure the columns and computed fields have a transparent background. The expression Mod(GetRow( ), 2) = 1 distinguishes odd rows from even rows:

```
If(Mod(GetRow(), 2) = 1, 16777215, 15790320)
```

This statement changes the expression in code:

```
dw_1.Object.rectangle_1.Brush.Color = &
       "0 ~t If(Mod(GetRow(), 2) = 1, 16777215,
15790320)"
```

Brush.Color and Brush.Hatch properties for a rectangle To highlight employees whose review date is approaching, draw a rectangle behind the row. This expression for the rectangle's Brush.Color property makes the rectangle light gray for employees for whom the month of the start date matches the current month or the next month:

```
If(month(start_date) = month(today())
or month(start_date) = month(today()) + 1
or (month(today()) = 12 and month(start_date) = 1),
12632256, 16777215)
```

A similar expression for the Brush.Hatch property makes the fill pattern of the rectangle Bdiagonal (1) for review dates that are approaching. Otherwise, the rectangle is transparent (7) so that it does not show:

```
If(month(start_date) = month(today())
or month(start_date) = month(today()) + 1
or (month(today()) = 12 and month(start_date) = 1),
1, 7)
```

You can also set the Pen.Color and Pen.Style properties to affect the outline of the rectangle.

If you wanted to change the Brush.Color property in code instead of setting it in the painter, the code would look like this:

```
dw_1.Object.rectangle_1.Brush.Color = &
       "'16777215 ~t " + &
       "If(month(start_date) = month(today()) " + &
       "or month(start_date) = month(today()) + 1 " + &
       "or (month(today()) = 12 " + &
       "and month(start_date) = 1), 12632256,
16777215)'"
```

Font.Height property for a rectangle This expression applied to the Font.Height property of a text control makes the text control in the first row of a DataWindow larger than it appears in other rows. Make sure the borders of the text control are large enough to accommodate the increased size:

```
If(GetRow() = 1, 500, 200)
```

This statement changes the expression for the text control t_desc in code:

```
dw_1.Object.t_desc.Font.Height = &
```

```
"200 ~t If(GetRow() = 1, 500, 200)"
```

For more information

For more information about DataWindow expressions, see Chapter 1, "DataWindow Operators and Expressions."

Nested strings and special characters for DataWindow object properties

The PowerScript escape character

DataWindow property values often involve specifying strings within strings. Embedded quotation marks need special treatment so that the strings are parsed correctly.

Tilde (~) is the escape character that allows you to nest quoted strings within other quoted strings and to specify special characters such as tabs and carriage returns. For DataWindow object properties, several levels of nested strings can create a complicated expression.

Techniques for quoting nested strings

Both double and single quotes are valid delimiters for strings. You can use this fact to simplify the specification of nested strings.

There are two ways to embed a string within another string. You can:

- Use the other type of quotation mark for the nested string. If the main string uses double quotes, the nested string can use single quotes.

  ```
  "If(state='MA',255,0)"
  ```

- Use the escape character to specify that a quote is part of the string instead the closure of a previous quote.

  ```
  "If(state=~"MA~",255,0)"
  ```

If the string includes a third level of nested strings, you need to add another tilde which must be accompanied by its own escape character, a second tilde. This is the reason that tildes are usually specified in odd numbers (1, 3, or 5 tildes).

This Modify expression (entered on a single line in code) shows three levels of nested strings:

```
dw_1.Modify(
     "DataWindow.Color = '255 ~t If(state=
~'MA~',255,0)'")
```

This version of the expression has more tildes because there are no single quotes:

```
dw_1.Modify("DataWindow.Color = ~"255 ~t If(state=
```

```
~~~"MA~~~",255,0)~"")
```

Common special characters

Strings can also include special characters, as shown in the previous example. This table lists the special characters that are most often used in DataWindow expressions.

Escape sequence	Meaning
~t	Tab
~r	Carriage return
~n	Newline or linefeed
~"	Double quote
~’	Single quote
~~	Tilde

A line break is a carriage return plus a newline (\r\n).

Special use of tilde

A special case of specifying tildes involves the EditMask.SpinRange property, whose value is two numbers separated by a tilde (not an escape character, simply a tilde). To specify this value in a script, you must use a nested string with four tildes, which is interpreted as a single tilde when parsed:

```
dw_1.Modify("benefits.EditMask.SpinRange='0~~~~10'")
```

More information

For more information about nested strings and special characters, see the *PowerScript Reference*.

Modify and Describe methods for properties

The following sections provide information about using Modify and Describe methods for DataWindow object properties:

- Advantage and drawbacks of Modify and Describe methods
- Handling errors from Modify and Describe methods

Advantage and drawbacks of Modify and Describe methods

Using the Describe and Modify methods to access DataWindow object property values has an advantage and some drawbacks. The examples here use Modify as illustrations, but similar considerations apply to Describe.

Advantage

Allows you to specify column and property names dynamically In your script, you can build a string that specifies the column and property names.

For example, the following code builds a string in which the default color value and the two color values in the If function are determined in the script. Notice how the single quotes around the expression are included in the first and last pieces of the string:

```
red_amount = Integer(sle_1.Text)
modstring = "emp_id.Color='" + &
       String(RGB(red_amount, 0, 0)) + &
       "~tIf(emp_status=~~'A~~'," + &
       String(RGB(255, 0, 0)) + &
       "," + &
       String(RGB(red_amount, 0, 0)) + &
       ")'"
Modify(modstring)
```

The resulting string when red_amount is set to 128 is:

```
emp_id.Color='128~tIf(emp_status=~'A~',255,128)'
```

The following is a simpler example without the If function. You do not need quotes around the value if you are not specifying an expression. Here the String and RGB functions result in a constant value in the resulting modstring:

```
Modify(ls_columnname + ".Color=" + &
       String(RGB(red_amount, 255, 255)))
```

Drawbacks

Setting several properties at once is possible but hard to debug Although you can set several properties in a single method call, it is harder to understand and debug scripts that do so.

For example, assume the following is entered on a single line in the script editor:

```
rtn = dw_1.Modify("emp_id.Font.Italic=0
oval_1.Background.Mode=0
oval_1.Background.Color=255")
```

Less efficient than an expression Using a DWObject variable in several property expressions is a little more efficient than setting several properties in a single call to Describe or Modify. However, if you want to be able to name controls dynamically, you might still choose to use Describe or Modify.

For examples of using a DWObject variable, see “Using the DWObject variable” on page 357.

Can require complex quoted strings When you specify an expression for a property value, it is difficult to specify nested quotes correctly—the code is hard to understand and prone to error. For Describe, this is less of a drawback—strings do not become as complex because they do not include an expression.

For example, this string entered on a single line in a script assigns a DataWindow expression to the Color property:

```
Modify("emp_id.Color=~"16777215 ~t
If(emp_status=~~~"A~~~",255,16777215)~"")
```

For more information about quoted strings, see "Nested strings and special characters for DataWindow object properties" on page 352.

Handling errors from Modify and Describe methods

Runtime errors do not occur when Describe and Modify try to access invalid controls or properties in the DataWindow object. The validity of the argument string is evaluated before the controls are accessed.

Modify

When the string that specifies the control and property to be accessed is invalid, Modify returns an error string, instead of the expected value, such as:

```
Line 1 Column 12: incorrect syntax.
```

You can use the error message to figure out what part of the string is incorrect. This is most useful when you are testing your scripts. The error message, which names the line and column number after which the string was not recognized, might not be helpful after your application is deployed.

Describe

When the string for Describe has an unrecognized property, Describe's return value ends with an exclamation point (!). Describe returns as many values as it recognizes up to the incorrect one.

When you specify a valid property but that property does not have a value (either because it has not been set or because its value is an expression that cannot be evaluated), Describe returns a question mark (?) for that property. The property's actual value is NULL.

Always check for errors
You should include error-checking code that checks for these return values. Other errors can occur later if you depend on settings that failed to take effect.

For more information

For more information on syntax and usage, see Describe and Modify in Chapter 9, "Methods for the DataWindow Control."

DataWindow property expressions

DataWindow property expressions use dot notation. These sections explain how to use the expressions and what syntax to use to construct them:

Basic structure of DataWindows and property expressions

Controls in a DataWindow

A DataWindow object is made up of many controls (such as Columns, Text, Pictures, and Reports). In PocketBuilder scripts, the datatype of these controls is DWObject. Each DWObject has a set of properties according to its type. The syntax of a property expression allows you to address any of these properties.

Object property

A DataWindow property expression uses the Object property of the DataWindow control to access the DataWindow object. Following the Object property, you specify a control name and one or more properties.

The simple syntax is:

dwcontrol.**Object**.*dwcontrolname*.*property*

For example:

```
dw_1.Object.empname.Resizeable
```

For the full syntax, see "Basic syntax for DataWindow property expressions" on page 364.

About DataWindow data expressions

Expressions that access data in a DataWindow object using dot notation use the Object and Data properties. These expressions are called **data expressions** (in contrast to property expressions); because of the intricate syntax for data expressions, they are described separately, in Chapter 4, "Accessing Data in Code."

Datatypes of DataWindow property expressions

DataWindow property values

The values of DataWindow object properties are strings. These strings can contain numeric or yes/no values, but the values you access are strings, not integers or boolean values.

Although the property values are really strings, the PowerScript compiler allows you to assign numbers and boolean values to properties whose strings represent numeric values or contain yes/no strings. This does not mean the datatype is integer or boolean. It is just a convenience when assigning a value to the property.

For example, both of these statements are correct:

```
dw_1.Object.empname.Border = 1
dw_1.Object.empname.Border = '1'
```

DataWindow property expressions

The datatype of a property expression is Any (not string), but the value of the data in the Any variable is a string. This may sound like an unnecessary distinction, but it does matter when you use a property expression as a method argument. If the method does not accept an Any variable as an argument, you might need to use the String function to cast the data to the correct datatype.

For example, because the MessageBox function accepts a string argument (not an Any datatype), the property expression is enclosed in a String conversion function:

```
MessageBox("Border", &
       String(dw_1.Object.empname.Border))
```

Using the DWObject variable

A PocketBuilder DWObject object is an object that exists within a DataWindow object. Each column, computed field, text control, or drawing control is a DWObject.

A DWObject reference allows you to refer directly to controls within a DataWindow.

You can use a DWObject variable to simplify DataWindow property and data expressions. A DWObject variable takes the place of several elements of the control's dot notation.

The following syntaxes and examples show how using a DWObject variable affects property and data expressions.

Property expressions

The simple syntax for a property expression is:

dwcontrol.Object.*dwcontrolname*.*property*

You can use a DWObject variable to refer to *dwcontrolname*.

Suppose that the code declares a DWObject variable and assigns the control within the DataWindow to the variable, using syntax like this:

DWObject *dwobjectvar*

dwobjectvar = *dwcontrol*.Object.*dwcontrolname*

The syntax of the expression itself becomes:

dwobjectvar.*property*

For example, if the DataWindow had a column named empname, a text control named t_emplabel, and a computed field named cf_average, you could make the following assignments:

```
DWObject dwo_column, dwo_text, dwo_compute
dwo_column = dw_1.Object.empname
dwo_text = dw_1.Object.t_emplabel
dwo_compute = dw_1.Object.cf_average
```

Data expressions

You can use a DWObject variable to refer to a column in a data expression. For example, this syntax gets data for a single row and column:

dwcontrol.Object.*columnname* {.*buffer* } {.*datasource* } [*rownum*]

Suppose that the code declares a DWObject variable and assigns the control within the DataWindow to the variable, using syntax like this:

DWObject *dwobjectvar*

dwobjectvar = *dwcontrol*.Object.*columnname*

The syntax of the expression itself becomes:

dwobjectvar. {.*buffer* } {.*datasource* } [*rownum*]

DWObject variables

You can get better performance by using a DWObject variable to resolve the object reference in a DataWindow property or data expression. Evaluating the reference once and reusing the resolved reference is more efficient than fully specifying the object reference again.

This technique yields the most benefit if your application uses compiled code or if you are using a DataWindow expression in a loop.

For example, this code is not optimized for best performance, because the fully specified data expression within the loop must be resolved during each pass:

```
integer li_data
FOR li_cnt = 1 to 100
       li_data = dw_1.Object.emp_salary[li_cnt]
       .. // Code to process data value
NEXT
```

This code has been optimized. The reference to the control within the DataWindow (emp_salary) is resolved once before the loop begins. The reference stored in the DWObject variable is reused repeatedly in the loop:

```
integer li_data
DWObject dwo_empsalary

dwo_empsalary = dw_1.Object.emp_salary

FOR li_cnt = 1 to 100
       li_data = dwo_empsalary.Primary[li_cnt]
       .. // Code to process data value
NEXT
```

Using a DWObject variable instead of a data expression

In a data expression for a column that refers to one item, the brackets for the row index identify the expression as a data expression (for information, see "Syntax for one or all data items in a named column" on page 333). However, if you assign the column control to a DWObject variable, the brackets incorrectly signify an array of objects. Therefore you must include a buffer name or data source to specify that you want data:

```
dw_1.Object.emp_salary[1] //Single data item

DWObject dwo_empsalary
dwo_empsalary = dw_1.Object.emp_salary
dwo_empsalary[1] // Incorrect: array of DWObject
dwo_empsalary.Primary[1] // Single data item
```

DWObject arguments for DataWindow events

Several DataWindow events pass a DWObject argument called dwo to the event script. The value is a resolved reference to a control within the DataWindow having something to do with the user's action that triggered the event. Often it is the column the user is changing or the control the user clicked.

What type of DWObject?

You can use DataWindow properties to find out more about the control stored in dwo. The first step is to find out the control's type so that subsequent statements will use properties that are appropriate for the control type. If an expression uses a property that does not correspond to the control's type, it will trigger the Error event. This statement in an event script gets the type:

```
ls_type = dwo.Type
```

The possible values that can be assigned to ls_type are:

bitmap (for Picture)
button
column
compute (for Computed Field)
graph
groupbox
line
ole
ellipse (for Oval)
rectangle
roundrectangle
report
tableblob
text
datawindow (*when the user doesn't click a specific control*)

You can write a CHOOSE CASE statement for the expected types.

After you have determined the type, you can get more details about the specific control.

Examples

If the control is a column, you can get the column name with this statement:

```
ls_name = dwo.Name
```

If the control is a column, you can get data from the whole column or from specific rows. You must specify the buffer from which you want to retrieve data. In this statement, row is another argument passed to the event so the value in ls_data is the data in the row and column the user clicked. In this example, if the column value is not a string, an error occurs (check ColType property to get the column datatype):

```
ls_data = dwo.Primary[row]
```

This statement assigns a new value to the row and column the user clicked. The assignment does not trigger the ItemChanged event and bypasses validation. If the column is not numeric, an error occurs:

```
dwo.Primary[row] = 41
```

This statement gets all the data in the column the user clicked. The data is stored as an array in the Any variable. An Any variable can hold all datatypes, so no error occurs:

```
Any la_data
la_data = dwo
```

This statement gets data in the column from selected rows. The data is stored as an array in the Any variable:

```
Any la_data
la_data = dwo.Selected
```

When a DataWindow property expression is evaluated

Expressions that refer to DataWindow object properties and data are not verified until your application runs.

No compiler checking

When your script is compiled, PocketBuilder does not verify the parameters of the expression that follow the Object property. Your application can select the DataWindow object in a DataWindow control during execution without invalidating the compiled script.

Potential execution errors

If the datatype of the expression is not compatible with how the expression is used, or if the specified rows or columns do not exist, then an error will occur during execution.

You can handle the error by surrounding the expression in a try-catch block or by writing a script for the DataWindow Error event.

Handling errors from DataWindow property expressions

What causes errors

An invalid DataWindow property expression causes a runtime error in your application. A runtime error causes the application to terminate unless you catch the error in a runtime error handler or unless there is a script for the Error event:

Table 5-3: Conditions that invalidate DataWindow property expressions

Conditions that cause errors	Possible causes
Invalid names of controls within the DataWindow object	Mistyping, which the compiler does not catch because it does not evaluate the expression. A different DataWindow object has been inserted in the control and it has different columns and controls.
A property is not valid for the specified control	Mistyping. The control is a different type than expected.

You can prevent the application from terminating by handling the error in the DataWindow control's Error event or by catching the error in a try-catch block.

Responding to errors in the Error event script

The Error event's arguments give you several options for responding to the error. You choose a course of action and set the *action* argument to a value of the ExceptionAction enumerated datatype.

ExceptionAction enumerated datatype
If you give the *action* argument a value other than ExceptionIgnore!, you will prevent error-handling code in try-catch blocks from executing. For more information on values for the ExceptionAction enumerated datatype, see the Error event description in the *PowerScript Reference*.

If you are trying to find out a property value and you know the expression might cause an error, you can include code that prepares for the error by storing a default value in an instance variable. Then the Error event script can return that value in place of the failed expression.

There are three elements to this technique: the declaration of an instance variable, the script that sets the variable's default value and then accesses a DataWindow property, and the Error event script. These elements are shown in Example 2 below.

Responding to errors in a try-catch block

You can prevent the application from terminating by handling the DataWindow runtime error (DWRuntimeError) in a try-catch block. If you are trying to find out a property value and you know the expression might cause an error, you can include code that automatically assigns a valid default value that can be substituted for the failed expression, as in Example 2 below.

Examples

Example 1 This code displays complete information about the error in a multilineedit mle_1.

The error event script:

```
mle_1.text = &
   "error#: " + string(errornumber) + "~r~n" + &
   "text: " + errortext + "~r~n" + &
   "parent: " + errorwindowmenu + "~r~n" + &
   "object: " + errorobject + "~r~n" + &
   "line: " + string(errorline) + "~r~n"
action = ExceptionIgnore!
```

The try-catch block:

```
Try
   ... //DataWindow property expression
Catch (DWRuntimeError myExc)
   mle_1.text = &
   "error#: " + string(myExc.number) + "~r~n" +&
   "text: " + myExc.text + "~r~n" + &
   "script: " + myExc.routinename + "~r~n" + &
   "object: " + myExc.objectname + "~r~n" + &
   "line: " + string(myExc.line) + "~r~n"
End Try
```

If the correct evaluation of the expression is not critical to the application, the application continues without terminating.

Example 2 This example provides a return value that will become the expression's value if evaluation of the expression causes an error.

There are three elements to code in the error event script. The instance variable is a string:

```
string is_dwvalue
```

This script for a button or other control stores a valid return value in an instance variable and then accesses a DataWindow property:

```
is_dwvalue = "5"
ls_border = dw_1.Object.id.Border
```

The Error event script uses the instance variable to provide a valid return value:

```
action = ExceptionSubstituteReturnValue!
returnvalue = is_dwvalue
```

The try-catch block:

```
try
   ls_border = dw_1.Object.id.Border
catch (DWRuntimeError myDWError)
   ls_border = "5"
```

```
end try
```

During execution, if the id column does not exist or some other error occurs, then the expression returns a valid border value—here the string "5". If you are using the Error event instead of a try-catch block, you must first store the value in an instance variable.

Basic syntax for DataWindow property expressions

Description

DataWindow property expressions use dot notation to specify the controls and properties that you want to access.

Syntax

dwcontrol.**Object**.*dwcontrolname* { .*property* } .*property* { = value }

Argument	Description
dwcontrol	The name of the DataWindow control or child DataWindow in which you want to get or set properties.
Object	Object indicates that subsequent elements refer to the DataWindow object within *dwcontrol*.
dwcontrolname	A control within the DataWindow object. Possible values are DataWindow (for properties that apply to the whole DataWindow) or the name of a column, computed field, graph, line, oval, picture, rectangle, roundrectangle, report, TableBlob, or text control.
property	A property that applies to *dwcontrolname*. If the property requires additional qualifying properties, list the additional properties, separating them with a dot. For lists of applicable properties, see the Property tables at the beginning of Chapter 3, “DataWindow Object Properties.”

Argument	Description
value	A string whose value is to be assigned to the property. If the property value is a number, *value* can either be a string whose value is a number or a numeric datatype. The value is stored as a string. If the property value is a yes or no value, *value* can be either a string whose value is "yes" or "no" or a boolean value (TRUE or FALSE). The value is stored as "yes" or "no" strings. If the property value can be an expression, then *value* can be a string that takes the form: *defaultvalue*~t *DataWindowexpression* where: • *Defaultvalue* is any value that is allowed for *property*. • *DataWindowexpression* is an expression that can include names of controls in the DataWindow and DataWindow expression functions. • *Defaultvalue* and *DataWindowexpression* are separated by a tab character (~t). For examples of DataWindow expressions, see "Using DataWindow expressions as property values" on page 349.

Datatype

Any. The datatype of the expression is Any, but actual data is a string.

For more information about the expression's datatype, see "Datatypes of DataWindow property expressions" on page 357.

Examples

Example 1 Boolean property values In this statement, the boolean value FALSE is stored as the string "no":

```
dw_1.Object.DataWindow.ReadOnly = FALSE
```

This statement displays the value of the ReadOnly property (either "yes" or "no") in the StaticText st_status:

```
st_status.Text = dw_1.Object.DataWindow.ReadOnly
```

When you test the value of a property in a relational expression, you must compare your test value to the stored values. For ReadOnly, stored values are yes or no, not boolean TRUE or FALSE:

```
IF dw_1.Object.DataWindow.Readonly = 'yes' THEN
```

This statement fails because the expression is not boolean:

```
IF dw_1.Object.DataWindow.Readonly THEN // Not valid
```

Example 2 Valid values for the Visible property are 0 and 1. You can set the property to numbers, yes and no, or TRUE and FALSE. Therefore, these three statements are equivalent:

```
dw_1.Object.street.Visible = FALSE
dw_1.Object.street.Visible = "NO"
dw_1.Object.street.Visible = 0
```

Example 3 This example tests whether the X property contains a constant (which can be converted to a number) or a DataWindow expression. The code assigns a default value of 50 to the variable li_x, which remains the value if the property contains an expression the script cannot convert:

```
integer li_x
IF IsNumber( dw_1.Object.id.X ) THEN
       li_x = Integer( dw_1.Object.id.X )
ELSE
       li_x = 50
END IF
```

Example 4 This script sets the X property to a DataWindow expression. The expression causes IDs with values less than 10 to be indented:

```
string modstring, ls_x
ls_x = "50"
modstring = ls_x + "~t" + &
       "If(id > 10, " + ls_x + "," + &
       String(li_x + 20 ) + ")"
dw_1.Object.id.X = modstring
```

Example 5 This example makes three columns updatable and reports the value of the Update property in the StaticText st_status. The reported value is "yes," not TRUE:

```
dw_1.Object.id.Update = TRUE
dw_1.Object.street.Update = TRUE
dw_1.Object.last_name.Update = TRUE

st_status.Text = &
       "Updateable: id " + dw_1.Object.id.Update + &
       ", street " + dw_1.Object.street.Update + &
       ", last_name " + dw_1.Object.last_name.Update
```

Example 6 This example checks whether the id column is set up as a spin control. If so, it sets the spin range to 0 through 10:

```
IF dw_1.Object.id.EditMask.Spin = "yes" THEN
       dw_1.Object.id.EditMask.SpinRange = "0~~~~10"
END IF
```

CHAPTER 6

DataWindow Constants

About this chapter

This chapter lists the PocketBuilder enumerated datatypes that provide constants for setting DataWindow property values.

Contents

About DataWindow constants

About constants

This section lists the constants that are defined in the DataWindow control for values of properties and arguments for methods. Constants have both a name and a numeric value.

Web DataWindow
Information in this chapter about the Web DataWindow does not apply to PocketBuilder applications. For more information about the Web DataWindow, see the *DataWindow Programmer's Guide* and the *Working with Web and JSP Targets* book in the PowerBuilder documentation set.

What values to use

PocketBuilder In PocketBuilder, constants are defined as sets of values associated with enumerated datatypes. Values for enumerated datatypes always end with an exclamation point. When an enumerated datatype is specified as the datatype, you must use the enumerated value. You cannot use the numeric equivalent.

```
dw1.BorderStyle = StyleRaised!
```

DataWindow object properties When setting DataWindow properties in PocketBuilder, you use the numeric value in quoted strings.

How this section is organized

This section lists the values according to the PocketBuilder enumerated datatypes, so you can see which values are available for setting a particular type of data. If you know a value's name but not the enumerated datatype it belongs to, you can find the value in the index of this book.

Alphabetical list of DataWindow constants

This section groups DataWindow constants according to enumerated datatype.

Alignment

Description

Values for specifying the alignment of text in DataWindow columns or text controls.

Values

Use the numeric values with the Alignment DataWindow object property.

PocketBuilder enumerated value	Numeric value	Meaning
Left!	0	Text is left aligned.
Right!	1	Text is right aligned.
Center!	2	Text is centered.
Justify!	3	Wrapped text is justified. The last line of text is not stretched to fill the area. So for a single line of text, justified alignment will appear to have no effect.

See also

Alignment

Band

Description

Values identifying the band containing the insertion point in a DataWindow control.

Values

PocketBuilder enumerated value	Web DataWindow	Numeric value	Meaning
Detail!	Detail	0	The detail band
Header!	Header	1	The header band
Footer!	Footer	2	The footer band

Border

Description

Values identifying the border style for a column in a DataWindow.

Used in the GetBorderStyle and SetBorderStyle methods and the Border property for DataWindow columns.

Values

PocketBuilder enumerated value	Numeric value	Meaning
NoBorder!	0	No border
ShadowBox!	1	Each data value is in a box that has a drop shadow
Box!	2	Each data value is surrounded by a rectangular border with no shading
ResizeBorder!	3	The column is resizable; the user can grab the border around any data value and drag it
Underline!	4	Each data value in the column is underlined
Lowered!	5	Each data value has a 3D border with shading to make it look lowered
Raised!	6	Each data value has a 3D border with shading to make it look raised

See also

Border
GetBorderStyle
SetBorderStyle

BorderStyle

Description

Values for specifying the border style of the DataWindow control.

Used for the Border property of the DataWindow control.

Values

PocketBuilder enumerated value	Numeric value	Meaning
StyleBox!	2	The DataWindow control is surrounded by a rectangular box without any shading
StyleLowered!	5	The control has a 3D border with shading to make it look lowered
StyleRaised!	6	The control has a 3D border with shading to make it look raised
StyleShadowBox!	1	The control has a rectangular border with a drop shadow

See also

Border

CharSet

Description

Values for specifying the character set used in the DataWindow.

Generally, the value for CharSet is derived from the font selected for controls within the DataWindow.

Values are used with the Font.CharSet DataWindow object property. Use the numeric values, not the enumerated values, for DataWindow object properties.

Values

PocketBuilder enumerated value	Numeric value	Meaning
—	1	The default character set for the specified font
CharSetAnsi!	0	Standard ANSI
CharSetUnicode!		Unicode
CharSetAnsiHebrew!		Right-to-left Hebrew
CharSetAnsiArabic!		Right-to-left Arabic
CharSetDBCS-Japanese!		Double-byte Japanese
—	2	Symbol
—	128	Shift-JIS
—	255	OEM

See also

Font.property

DWBuffer

Description

Values for specifying the DataWindow buffer containing the rows you want to access.

Used in many DataWindow methods that access data.

Values

PocketBuilder enumerated value	Web DataWindow	Numeric value	Meaning
Primary!	Primary	0	The data in the primary buffer, meaning data that has not been deleted or filtered out. (Default value when argument is optional.)
Delete!	Delete	1	Data in the delete buffer, meaning data that has been deleted from the DataWindow but has not been committed to the database.
Filter!	Filter	2	Data in the filter buffer, meaning data that has been removed from view.

See also

GetItemStatus
SetItem

DWConflictResolution

Description

Values for specifying how to handle potential conflicts when synchronizing DataWindows in a distributed application.

Values

PocketBuilder enumerated value	Numeric value	Meaning
FailOnAnyConflict!	0	Prevents changes from being synchronized if data in the source DataWindow has changed since its state was captured. (Default value when argument is optional.)
AllowPartialChanges!	1	Allows changes that are not in conflict to be applied.

See also

SetChanges on page 600 explains how to test whether conflicts exist.

DWItemStatus

Description

Values for specifying how DataWindow data will be updated in the database.

Values

PocketBuilder enumerated value	Web DataWindow	Numeric value	Meaning
NotModified!	NotModified	0	The information in the row or column is unchanged from what was retrieved.
DataModified!	DataModified	1	The information in the column or one of the columns in the row has changed since it was retrieved.
New!	New	2	The row is new but no values have been specified for its columns. (Applies to rows only, not to individual columns.)
NewModified!	NewModified	3	The row is new, and values have been assigned to its columns. In addition to changes caused by user entry or the SetItem method, a new row gets the status NewModified when one of its columns has a default value. (Applies to rows only, not to individual columns.)

See also

SetItemStatus on page 612 describes how to change individual item statuses and how the status affects the SQL statements that update the database.

FillPattern

Description

Values for the fill pattern of shapes (for example, bars or pie slices) in a graph control.

Used in Get/SetSeriesStyle and Get/SetDataStyle methods for graph controls in a DataWindow or for PocketBuilder graph controls.

Values

PocketBuilder enumerated value	Numeric value	Meaning
Solid!	0	A solid color
Horizontal!	1	Horizontal lines
Vertical!	2	Vertical lines
FDiagonal!	3	Lines from upper left to lower right
BDiagonal!	4	Lines from lower left to upper right
Square!	5	A pattern of squares
Diamond!	6	A pattern of diamonds

See also

GetDataStyle
GetSeriesStyle
SetDataStyle
SetSeriesStyle

grColorType

Description

Values for specifying the purpose of a color in a graph, for example, background or foreground.

Used in Get/SetSeriesStyle and Get/SetDataStyle methods for graph controls in a DataWindow or for PocketBuilder graph controls.

Values

PocketBuilder enumerated value	Numeric value	Meaning
Foreground!	0	Text (fill color)
Background!	1	The background color
Shade!	2	The shaded area of three-dimensional graphics
LineColor!	3	The color of the line.

See also

GetDataStyle
GetSeriesStyle
SetDataStyle
SetSeriesStyle

grDataType

Description

Values for specifying X or Y value when getting information about a scatter graph.

Used in the GetData method for graph controls in a DataWindow or for PocketBuilder graph controls.

Values

PocketBuilder enumerated value	Numeric value	Meaning
yValue!	1	(Default) The y value of the data point
xValue!	0	The x value of the data point

See also

GetData

grObjectType

Description

Values that identify parts of a graph.

Used as the return value of the ObjectAtPointer method for graph controls in a DataWindow or for PocketBuilder graph controls.

Values

PocketBuilder enumerated value	Numeric value	Meaning
TypeGraph!	0	Any place within the graph control that is not another grObjectType
TypeTitle!	4	The title of the graph
TypeLegend!	8	Within the legend box, but not on a series label
TypeData!	2	A data point or other data marker
TypeCategory!	3	A label for a category
TypeCategoryAxis!	10	The category axis or between the category labels
TypeCategoryLabel!	6	The label of the category axis
TypeSeries!	1	The line that connects the data points of a series when the graph's type is line or on the series label in the legend box
TypeSeriesAxis!	9	The series axis of a 3D graph
TypeSeriesLabel!	5	The label of the series axis of a 3D graph.

PocketBuilder enumerated value	Numeric value	Meaning
TypeValueAxis!	11	The value axis, including on the value labels
TypeValueLabel!	7	The user clicked the label of the value axis

See also

ObjectAtPointer

grSymbolType

Description

Values for the symbols associated with data points in a graph.

Used in Get/SetSeriesStyle and Get/SetDataStyle methods for graph controls in a DataWindow or for PocketBuilder graph controls.

Values

PocketBuilder enumerated value	Numeric value	Meaning
NoSymbol!	0	None
SymbolHollowBox!	1	A hollow box
SymbolX!	2	An X
SymbolStar!	3	A star
SymbolHollowUpArrow!	4	An outlined up arrow
SymbolHollowDownArrow!	5	An outlined down arrow
SymbolHollowCircle!	6	An outlined circle
SymbolHollowDiamond!	7	An outlined diamond
SymbolSolidBox!	8	A filled box
SymbolSolidDownArrow!	9	A filled down arrow
SymbolSolidUpArrow!	10	A filled up arrow
SymbolSolidDiamond!	11	A filled diamond
SymbolSolidCircle!	12	A filled circle
SymbolPlus!	13	A plus sign

See also

GetDataStyle
GetSeriesStyle
SetDataStyle
SetSeriesStyle

LineStyle

Description

Values for the pattern of lines in a graph.

PocketBuilder
In PocketBuilder, all dashed and dotted line styles are represented as dashed lines.

Used in Get/SetSeriesStyle and Get/SetDataStyle methods for graph controls in a DataWindow or for PocketBuilder graph controls.

Values

PocketBuilder enumerated value	Numeric value	Meaning
Continuous!	0	The line style is a solid line
Dash!	1	The line style is ----
DashDot!	2	The line style is -.-.-.
DashDotDot!	3	The line style is -..-..-..
Dot!	4	The line style is .....
Transparent!	5	The line allows the background shapes to show through

See also

GetDataStyle
GetSeriesStyle
SetDataStyle
SetSeriesStyle

RowFocusInd

Description

Values for specifying the indicator for the current row in a DataWindow.

Used in the SetRowFocusIndicator method for DataWindow controls.

Values

PocketBuilder enumerated value	Numeric value	Meaning
Off!	0	There is no indicator for the current row
FocusRect!	1	The row with focus has a dotted rectangle around it
Hand!	2	A pointing hand appears in the left margin of the DataWindow beside the row with focus

See also

SetRowFocusIndicator

SaveAsType

Description

Values for specifying a format for data you want to save.

Used in the SaveAs method for saving the data of a DataWindow, a graph control in a DataWindow, or a PocketBuilder graph control.

Values

PocketBuilder enumerated value	Web DataWindow	Numeric value	Meaning
Excel!	Excel	0	Microsoft Excel format
Text!	Text	1	(Default) Tab-separated columns with a return at the end of each row
CSV!	CSV	2	Comma-separated values
WK1!	WK1	5	Lotus 1-2-3 format
DIF!	DIF	6	Data Interchange Format
SQLInsert!	SQLInsert	9	SQL syntax
HTMLTable!	HTMLTable	13	HTML TABLE, TR, and TD elements
Excel5!	Excel5	14	Microsoft Excel Version 5 format

See also

SaveAs

SQLPreviewFunction

Description

Values passed to the SQLPreview DataWindow event to indicate what method triggered the event.

Values

PocketBuilder enumerated value	Numeric value	Meaning
PreviewFunction Retrieve!	1	The program called the DataWindow Retrieve method
PreviewFunction ReselectRow!	2	The program called the DataWindow ReselectRow method
PreviewFunction Update!	3	The program called the Datawindow Update method

See also

SQLPreview

SQLPreviewType

Description

Values passed to the SQLPreview DataWindow event to indicate what SQL statement is being sent to the DBMS.

Values

PocketBuilder enumerated value	Numeric value	Meaning
PreviewSelect!	1	A SELECT statement
PreviewInsert!	2	An INSERT statement
PreviewDelete!	3	A DELETE statement
PreviewUpdate!	4	An UPDATE statement

See also

SQLPreview

CHAPTER 7

Properties of the DataWindow Control and DataStore

About this chapter

The chapter lists the properties of the DataWindow control and DataStore. These properties can be set in code to control the appearance and behavior of the container for the DataWindow object.

Properties for the PocketBuilder DataWindow

These properties are also documented in the PocketBuilder book *Objects and Controls*.

Properties for DataStore objects

You can set properties of a DataStore object in code using dot notation.

Table 7-1: Setting DataStore properties using dot notation

DataStore property	Datatype	Description
DataObject	String	Specifies the name of the DataWindow or Report object associated with the control.
ClassDefinition	PowerObject	An object of type PowerObject containing information about the class definition of the object or control.
Object	DWObject	Used for the direct manipulation of controls within a DataWindow object from a script. These controls could be, for example, columns or text controls. For information, see Chapter 4, "Accessing Data in Code" and Chapter 5, "Accessing DataWindow Object Properties in Code."

Properties for DataWindow controls

You can set properties of a DataWindow control in the window or user object painter or in code.

Table 7-2: Properties of DataWindow controls

DataWindow property	Datatype	Description
Border	Boolean	Specifies whether the control has a border. Values are: • True — Control has a border. • False — Control does not have a border.
BorderStyle	BorderStyle (enumerated)	Specifies the border style of the control. Values are: StyleBox! StyleLowered! StyleRaised! StyleShadowBox!
BringToTop	Boolean	Specifies whether PocketBuilder moves the control to the top of the front-to-back order.
ClassDefinition	PowerObject	An object of type PowerObject containing information about the class definition of the object or control.
ControlMenu	Boolean	Specifies whether the Control Menu box displays in the control title bar. Values are: • True — Control Menu box displays in the control title bar. • False — Control Menu box does not display in the control title bar.
DataObject	String	Specifies the name of the DataWindow object or Report object associated with the control.
DragAuto	Boolean	Specifies whether PocketBuilder puts the control automatically into Drag Mode. DragAuto has these boolean values: • True — When the control is clicked, the control is automatically in Drag Mode. • False — When the control is clicked, the control is not automatically in Drag Mode. You have to manually put the control into Drag Mode by using the Drag function.

DataWindow property	Datatype	Description
DragIcon	String	Specifies the name of the stock icon or the file containing the icon you want to display when the user drags the control (the ICO file). The default icon is a box the size of the control. When the user drags the control, the icon displays when the control is over an area in which the control can be dropped (a valid drop area). When the control is over an area that is not a valid drop area, the No-Drop icon displays.
Enabled	Boolean	Specifies whether the control is enabled (can be selected). Values are: • True — Control is enabled. • False — Control is not enabled.
Height	Integer	Specifies the height of the DataWindow control, in PowerBuilder units.
HScrollBar	Boolean	Specifies whether a horizontal scroll bar displays in the control when all the data cannot be displayed at one time. Values are: • True — Horizontal scroll bar is displayed. • False — Horizontal scroll bar is not displayed.
HSplitScroll	Boolean	Specifies whether the split bar displays in the control. Values are: • True — Split bar is displayed. • False — Split bar is not displayed.
Icon	String	Specifies the name of the ICO file that contains the icon that displays when the DataWindow control is minimized.
LiveScroll	Boolean	Scrolls the rows in the DataWindow control while the user is moving the scroll box.
MaxBox	Boolean	Specifies whether a Maximize Box displays in the DataWindow control title bar. Values are: • True — Maximize Box displays. • False — Maximize Box does not display.
MinBox	Boolean	Specifies whether a Minimize Box displays in the DataWindow control title bar. Values are: • True — Minimize Box displays. • False — Minimize Box does not display.

DataWindow property	Datatype	Description
Object	DWObject	Used for the direct manipulation of controls within a DataWindow object from a script. These controls could be, for example, columns or text controls. For information, see Chapter 4, “Accessing Data in Code” and Chapter 5, “Accessing DataWindow Object Properties in Code.”
Resizable	Boolean	Specifies whether the DataWindow control is resizeable. Values are: • True — DataWindow is resizeable. • False — DataWindow is not resizeable.
RightToLeft	Boolean	Not supported in PocketBuilder. Specifies that characters should be displayed in right-to-left order. The application must be running on a Hebrew or Arabic version of PowerBuilder under an operating system that supports right-to-left display. Values are: • True — Characters display in right-to-left order. • False — Characters display in left-to-right order.
TabOrder	Integer	Specifies the tab value of the DataWindow control within the window or user object. (0 means the user cannot tab to the control.)
Tag	String	Specifies the tag value assigned to the DataWindow control.
Title	String	Specifies the text that displays in the DataWindow control title bar.
TitleBar	Boolean	Specifies whether a title bar displays in the DataWindow control. The user can move the DataWindow control only if it has a title bar. Values are: • True — Title bar is displayed in control. • False — No title bar is displayed in control.
Visible	Boolean	Specifies whether the DataWindow control is visible. Values are: • True — Control is visible. • False — Control is not visible.

DataWindow property	Datatype	Description
VScrollBar	Boolean	Specifies whether a vertical scroll bar displays in the control when not all the data can be displayed at one time. Values are: • True — Vertical scroll bar is displayed. • False — Vertical scroll bar is not displayed.
Width	Integer	Specifies the width of the DataWindow control, in PowerBuilder units.
X	Integer	Specifies the X position (the distance from the left edge of the window), in PowerBuilder units.
Y	Integer	Specifies the Y position (the distance from the top edge of the window), in PowerBuilder units.

CHAPTER 8

DataWindow Events

About this chapter

This chapter describes what DataWindow objects are and the ways you can use them in various programming environments.

Contents

About return values for DataWindow events

Use a RETURN statement as the last statement in the event script. The datatype of the value is long.

For example, in the ItemChanged event, set the return code to 2 to reject an empty string as a data value:

```
IF data = "" THEN
       RETURN 2
```

Categories of DataWindow events

The reference entries are listed in alphabetical order. To help you find the event you need, the events are organized here by the type of actions that trigger them.

Changing data

EditChanged
ItemChanged
ItemError
DropDown for drop-down lists

Database access

DBError

	RetrieveStart RetrieveRow RetrieveEnd SQLPreview UpdateStart UpdateEnd
Error handling	DBError Error ItemError
Focus	GetFocus LoseFocus ItemFocusChanged RowFocusChanging RowFocusChanged
Key presses	KeyDown ProcessEnter TabOut BackTabOut TabDownOut TabUpOut
Mouse actions	ButtonClicked ButtonClicking Clicked DoubleClicked DragDrop DragEnter DragLeave DragWithin MouseMove MouseUp RButtonDown
Printing	PrintStart PrintPage PrintMarginChange PrintEnd
Scrolling	ScrollHorizontal ScrollVertical
Miscellaneous	Constructor Destructor

Resize
GraphCreate for Graph controls and presentation styles
HTMLContextApplied for Web DataWindow
MessageText for crosstab DataWindows

Alphabetical list of DataWindow events

The list of DataWindow events follows in alphabetical order.

BackTabOut

Description

Occurs when the user presses Shift+Tab or, in some edit styles, the left arrow, to move focus to the prior cell in the DataWindow.

PocketBuilder	✗
PowerBuilder	✓

PowerBuilder event information

Event ID: pbm_dwnbacktabout

Return codes

There are no special outcomes for this event. The only code is:

0 Continue processing

ButtonClicked

Description

Occurs when the user clicks a button inside a DataWindow object.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

PocketBuilder event information

Event ID: pbm_dwnbuttonclicked

Argument	Description
row	Long by value. The number of the row the user clicked.
actionreturncode	Long by value. The value returned by the action performed by the button. For information about return values, see the Action DataWindow object property.
dwo	DWObject by value. A reference to the control within the DataWindow under the pointer when the user clicked.

Return codes

There are no special outcomes for this event. The only code is:

0 Continue processing

Usage

The ButtonClicked event executes code after the action assigned to the button has occurred. This event is fired only if you have not selected Suppress Event Processing for the button. If Suppress Event Processing is on, only the Clicked event and the action assigned to the button are executed when the button is clicked.

If Suppress Event Processing is off, the Clicked event and the ButtonClicked event are fired. If the return code of the ButtonClicking event is 0, the action assigned to the button is executed and the ButtonClicked event is fired. If the return code of the ButtonClicking event is 1, neither the action nor the ButtonClicked event are executed.

Examples

This statement in the ButtonClicked event displays the value returned by the button's action:

```
MessageBox(" ", actionreturncode)
```

This statement in the ButtonClicked event displays the value returned by the button's action:

```
Stringls_Object
String       ls_Win

ls_Object = String(dwo.name)
If ls_Object = "cb_close" Then
    Close(Parent)
ElseIf ls_Object = "cb_help" Then
    ls_win = parent.ClassName()
    f_open_help(ls_win)
End If
```

See also

ButtonClicking

ButtonClicking

Description

Occurs when the user clicks a button. This event occurs before the ButtonClicked event.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

PocketBuilder event information
Event ID: pbm_dwnbuttonclicking

Argument	Description
row	Long by value. The number of the row the user clicked.
dwo	DWObject by value. A reference to the control within the DataWindow under the pointer when the user clicked.

Return codes

Set the return code to affect the outcome of the event:

0 Execute the action assigned to the button, then trigger the ButtonClicked event

1 Prevent the action assigned to button from executing and the ButtonClicked event from firing

For information on setting the return code in a particular environment, see "About return values for DataWindow events" on page 387.

Usage

Use the ButtonClicking event to execute code before the action assigned to the button occurs. If the return code is 0, the action assigned to the button is then executed and the ButtonClicked event is fired. If the return code is 1, the action and the ButtonClicked event are inhibited.

This event is fired only if you have not selected Suppress Event Processing for the button.

The Clicked event is fired before the ButtonClicking event.

Examples

This statement in the ButtonClicking event displays a message box before proceeding with the action assigned to the button:

```
MessageBox(" ", "Are you sure you want to proceed?")
```

See also

ButtonClicked

Clicked

Description

Occurs when the user clicks anywhere in a DataWindow control.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

PocketBuilder event information

Event ID: pbm_dwnlbuttonclk

Argument	Description
xpos	Integer by value. The distance of the pointer from the left side of the DataWindow workspace. The distance is given in pixels.
ypos	Integer by value. The distance of the pointer from the top of the DataWindow workspace. The distance is given in pixels.
row	Long by value. The number of the row the user clicked. If the user does not click on a row, the value of the row argument is 0. For example, row is 0 when the user clicks outside the data area, or in the header, summary, or footer area.
dwo	DWObject by value. A reference to the control within the DataWindow under the pointer when the user clicked.

Return codes

Set the return code to affect the outcome of the event:

0 Continue processing
1 Prevent the focus from changing

For information on setting the return code in a particular environment, see "About return values for DataWindow events" on page 387.

Usage

The dwo, Name, or object argument provides easy access to the control the user clicks within the DataWindow. You do not need to know the coordinates of elements within the DataWindow to program control-specific responses to the user's clicks. For example, you can prevent editing of a column and use the Clicked script to set data or properties for the column and row the user clicks.

A click can also trigger RowFocusChanged and ItemFocusChanged events. A double-click triggers a Clicked event, then a DoubleClicked event.

The xpos and ypos arguments provide the same values the functions PointerX and PointerY return when you call them for the DataWindow control. For graphs in DataWindow controls, the ObjectAtPointer method provides similar information about objects within the graph control.

Examples

This code highlights the row the user clicked.

```
This.SelectRow(row, TRUE)
```

If the user clicks on a column heading, this code changes the color of the label and sorts the associated column. The column name is assumed to be the name of the heading text control without _t as a suffix.

```
string ls_name
```

```
IF dwo.Type = "text" THEN
       dwo.Color = RGB(255,0,0)

       ls_name = dwo.Name
       ls_name = Left(ls_name, Len(ls_name) - 2)

       This.SetSort(ls_name + ", A")
       This.Sort()
END IF
```

See also

ButtonClicked
ButtonClicking
DoubleClicked
ItemFocusChanged
RButtonDown
RowFocusChanged
RowFocusChanging

Constructor

Description

Occurs when the DataWindow control or DataStore object is created, just before the Open event for the window that contains the control.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

PocketBuilder event information
Event ID: pbm_constructor

Return codes

There are no special outcomes for this event. The only code is:

0 Continue processing

Usage

You can write code for the Constructor event to affect DataWindow properties before it is displayed.

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

DBError

Description

Occurs when a database error occurs in the DataWindow or DataStore.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

PocketBuilder event information

Event ID: pbm_dwndberror

Argument	Description
sqldbcode	Long by value. A database-specific error code. See your DBMS documentation for information on the meaning of the code. When there is no error code from the DBMS, sqldbcode contains one of these values: -1 — Cannot connect to the database because of missing values in the transaction object. -2 — Cannot connect to the database. -3 — The key specified in an Update or Retrieve no longer matches an existing row. This can happen when another user has changed the row after you retrieved it. -4 — Writing a blob to the database failed.
sqlerrtext	String by value. A database-specific error message.
sqlsyntax	String by value. The full text of the SQL statement being sent to the DBMS when the error occurred.
buffer	DWBuffer by value. The buffer containing the row involved in the database activity that caused the error. For a list of valid values, see DWBuffer on page 372.
row	Long by value. The number of the row involved in the database activity that caused the error (the row being updated, selected, inserted, or deleted).

Return codes

Set the return code to affect the outcome of the event:

0 Display the error message
1 Do not display the error message

For information on setting the return code in a particular environment, see “About return values for DataWindow events” on page 387.

Usage

By default, when the DBError event occurs in a DataWindow control, it displays a system error message. You can display your own message and suppress the system message by specifying a return code of 1 in the DBError event.

Since DataStores are nonvisual, a system message does not display when the DBError event occurs in a DataStore. You must add code to the DBError event to handle the error.

If the row that caused the error is in the Filter buffer, you must unfilter it if you want the user to correct the problem.

Reported row number
The row number stored in row is the number of the row in the buffer, not the number the row had when it was retrieved into the DataWindow object.

Examples

This example illustrates how to display custom error messages for particular database error codes:

```
CHOOSE CASE sqldbcode

       CASE -195 // Required value is NULL.
       MessageBox("Database Problem", &
          "Error inserting row " + string(row) &
          + ". Please specify a value for Employee ID.")
       CASE ...
       // Code to handle other errors

END CHOOSE

RETURN 1 // Do not display system error message
```

See also

Error

Destructor

Description

Occurs when the DataWindow control or DataStore object is destroyed, immediately after the Close event of a window or form.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

PocketBuilder event information
Event ID: pbm_destructor

Return codes

There are no special outcomes for this event. The only code is:

0 Continue processing

Usage

The Destructor event destroys the DataWindow control or DataStore object and removes it from memory. After it has been destroyed, you can no longer refer to it in other event code. (If you do, a runtime error occurs.)

Restriction on methods
Calling a DataStore method that accesses the underlying DataStore internals within this event is not a valid coding practice and can fail silently. Such methods include RowCount, DBCancel, and Modify.

When you issue a DESTROY on a DataStore, the Destructor event is triggered and a Windows WM_DESTROY message is added to the object's message queue. WM_DESTROY invalidates the memory for the DataStore. If the WM_DESTROY message is handled before the method calls in the Destructor event, methods that attempt to access the destroyed memory fail silently.

See also

Constructor

DoubleClicked

Description

Occurs when the user double-clicks in a DataWindow control.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

PocketBuilder event information
Event ID: pbm_dwnlbuttondblclk

Argument	Description
xpos	Integer by value. The distance of the pointer from the left side of the DataWindow's workspace. The distance is given in pixels.
ypos	Integer by value. The distance of the pointer from the top of the DataWindow's workspace. The distance is given in pixels.

Argument	Description
row	Long by value. The number of the row the user double-clicked. If the user did not double-click on a row, the value of the row argument is 0. For example, row is 0 when the user double-clicks outside the data area, in text or spaces between rows, or in the header, summary, or footer area.
dwo	DWObject by value. A reference to the control within the DataWindow the user double-clicked.

Return codes

There are no special outcomes for this event. The only code is:

0 Continue processing

Usage

The dwo, Name, or DWObject argument provides easy access to the control the user clicks. You do not need to know the coordinates of elements within the DataWindow to program control-specific responses to the user's clicks. For example, you can prevent editing of a column and use the Clicked event to set data or properties for the column and row the user clicks.

The xpos and ypos arguments provide the same values the functions PointerX and PointerY return when you call them for the DataWindow control.

Examples

This example displays a message box reporting the row and column clicked and the position of the pointer relative to the upper-left corner of the DataWindow control:

```
string ls_columnname

IF dwo.Type = "column" THEN
       ls_columnname = dwo.Name
END IF

MessageBox("DoubleClicked Event", &
       "Row number: " + row &
       + "~rColumn name: " + ls_columnname &
       + "~rDistance from top of dw: " + ypos &
       + "~rDistance from left side of dw: " + xpos)
```

See also

Clicked
ItemFocusChanged
RButtonDown
RowFocusChanged
RowFocusChanging

DragDrop

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	✓

PocketBuilder event information
Event ID: pbm_dwndragdrop

Argument	Description
source	DragObject by value. A reference to the control being dragged.
row	Long by value. The number of the row the pointer was over when the user dropped the object. If the pointer was not over a row, the value of the row argument is 0. For example, row is 0 when the pointer is outside the data area, in text or spaces between rows, or in the header, summary, or footer area.
dwo	DWObject by value. A reference to the control under the pointer within the DataWindow when the user dropped the object.

Return codes

There are no special outcomes for this event. The only code is:

0 Continue processing

Examples

PocketBuilder This example for the DragDrop event for a DataWindow checks whether the source object is a DataWindow control. If so, it finds out the current row in the source and moves it to the target:

```
DataWindow ldw_Source

IF source.TypeOf() = DataWindow! THEN
       ldw_Source = source
       IF row > 0 THEN
           ldw_Source.RowsMove(row, row, Primary!, &
              This, 1, Primary!)
       END IF
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	✓

PocketBuilder event information

Event ID: pbm_dwndragenter

Argument	Description
source	DragObject by value. A reference to the control being dragged.

Return codes

There are no special outcomes for this event. The only code is:

0 Continue processing

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	✓

PocketBuilder event information

Event ID: pbm_dwndragleave

Argument	Description
source	DragObject by value. A reference to the control being dragged.

Return codes

There are no special outcomes for this event. The only code is:

0 Continue processing

Examples

This example checks the name of the control being dragged and if it is dw_1, it cancels the drag operation:

```
IF ClassName(source) = "dw_1" THEN
      dw_1.Drag(Cancel!)
END If
```

See also

DragDrop
DragEnter
DragWithin

DragWithin

Description

Occurs when the user is dragging an object within the control.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✓

PocketBuilder event information

Event ID: pbm_dwndragleave

Argument	Description
source	DragObject by value. A reference to the control being dragged.
row	Long by value. The number of the row the pointer is over. If the pointer is not over a row, the value of the row argument is 0. For example, row is 0 when the pointer is outside the data area, in text or spaces between rows, or in the header, summary, or footer area.
dwo	DWObject by value. A reference to the control under the pointer within the DataWindow.

Return codes

There are no special outcomes for this event. The only code is:

0 Continue processing

Usage

The DragWithin event occurs repeatedly as the mouse moves within the control.

See also

DragDrop
DragEnter
DragLeave

DropDown

Description

Occurs just before the list provided by a DropDownDataWindow is displayed. Use this event to retrieve new data for the child DataWindow.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

A DropDownDataWindow is a drop-down choice list whose data is provided by retrieving data for another DataWindow. To create a DropDownDataWindow, you assign the DropDownDataWindow edit style to a column and associate it with another DataWindow that retrieves the data for the choice list.

PocketBuilder event information

Event ID: pbm_dwndropdown

DropDown is not a standard PocketBuilder DataWindow event. To write a script for this event, you must first define a user-defined event for the event ID pbm_dwndropdown.

Return codes

There are no special outcomes for this event. The only code is:

0 Continue processing

EditChanged

Description

Occurs for each keystroke the user types in an edit control in the DataWindow.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

PocketBuilder event information

Event ID: pbm_dwnchanging

Argument	Description
row	Long by value. The number of the row containing the item whose value is being changed.

Argument	Description
dwo	DWObject by value. A reference to the column containing the item whose value is being changed. Dwo is a reference to the column control, not the name of the column.
data	String by value. The current contents of the DataWindow edit control.

Return codes

There are no special outcomes for this event. The only code is:

0 Continue processing

Examples

This example displays the row and column that the user is editing in a StaticText control:

```
st_1.Text = "Row " + String(row) &
       + " in column " + dwo.Name
```

See also

ItemChanged

Error

Description

Occurs when an error is found in a data or property expression for an external object or a DataWindow object.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

PocketBuilder event information

Event ID: None

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.

Argument	Description
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. This is the default action. The error condition triggers the SystemError event if you do not handle the error in a Try-Catch block. • 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.)

Usage

DataWindow objects are dynamic. Expressions that use dot notation to refer to data and properties of these objects might be valid under some runtime conditions but not others. The Error event allows you to respond to this dynamic situation with error recovery logic.

The Error event also allows you to respond to communications errors in the client component of a distributed application. In the Error event for a custom connection object, you can tell PocketBuilder what action to take when an error occurs during communications between the client and the server.

The Error event gives you an opportunity to substitute a default value when the error is not critical to your application. Its arguments also provide information that is helpful in debugging. For example, the arguments can help you debug DataWindow data expressions that cannot be checked by the compiler—such expressions can only be evaluated during execution.

When to substitute a return value
The ExceptionSubstituteReturnValue! action allows you to substitute a return value when the last element of an expression causes an error. Do not use ExceptionSubstituteReturnValue! to substitute a return value when an element in the middle of an expression causes an error. The substituted return value will not match the datatype of the unresolved object reference and will cause a system error.

The ExceptionSubstituteReturnValue! action is most useful for handling errors in data expressions.

For DataWindows, if an error occurs while evaluating a data or property expression, error processing occurs like this:

1 The Error event occurs.

   If you use a Try-Catch block, it is best not to script the Error event.

2 If the Error event has no script or its action argument is not changed from the default action (ExceptionFail!), either a catch statement is executed or the SystemError event occurs.

3 If you do not handle the error in a Try-Catch block and the SystemError event has no script, an application error occurs and the application is terminated.

The chapter on "Using DataWindow Objects" in the *Resource Guide* (or the *DataWindow Programmer's Guide* in the PowerBuilder documentation set) contains a table of correspondences between Error event arguments and DWRuntimeError properties. You can use the DWRuntimeError properties in a Try-Catch block to obtain the same information about an error condition that you would otherwise obtain from Error event arguments.

For information about using data and property expressions for DataWindow objects, see Chapter 4, "Accessing Data in Code," and Chapter 5, "Accessing DataWindow Object Properties in Code."

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

GetFocus

Description

Occurs just before the control receives focus (before it is selected and becomes active).

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

PocketBuilder event information
Event ID: pbm_dwnsetfocus

Return codes

There are no special outcomes for this event. The only code is:

0 Continue processing

See also

Clicked
LoseFocus

GraphCreate

Description

Occurs after the DataWindow control creates a graph and populates it with data, but before it has displayed the graph. In this event, you can change the appearance of the data about to be displayed.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

PocketBuilder event information
Event ID: pbm_dwngraphcreate

GraphCreate is not a standard PocketBuilder DataWindow event. To write a script for this event, you must first define a user-defined event for the event ID pbm_dwngraphcreate.

Return codes

There are no special outcomes for this event. The only code is:

0 Continue processing

Examples

The following statement sets to black the foreground (fill) color of the Q1 series in the graph gr_quarter, which is in the DataWindow control dw_report. The statement is in the user event GraphCreate, which is associated with the event ID pbm_dwngraphcreate:

```
dw_report.SetSeriesStyle("gr_quarter", "Q1",  &
       foreground!, 0)
```

See also

GetFocus

HTMLContextApplied

Description

Occurs when the SetHTMLAction method has been called to apply an action to a DataWindow control or DataStore. The event occurs after the context has been set but before the action is applied.

PocketBuilder	✗
PowerBuilder	✓

PowerBuilder event information

Event ID: pbm_dwnhtmlcontextapplied

Return codes

Set the return code to affect the outcome of the event:

0 Continue processing (execute the action)
1 Prevent the action from being applied

ItemChanged

Description

Occurs when a field in a DataWindow control has been modified and loses focus (for example, the user presses ENTER, the TAB key, or an arrow key or clicks the mouse on another field within the DataWindow). It occurs before the change is applied to the item. ItemChanged can also occur when the AcceptText or Update function is called for a DataWindow control or DataStore object.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

PocketBuilder event information

Event ID: pbm_dwnitemchange

Argument	Description
row	Long by value. The number of the row containing the item whose value is being changed.
dwo	DWObject by value. A reference to the column containing the item whose value has been changed. Dwo is a reference to the column control, not the name of the column.
data	String by value. The new data the user has specified for the item.

Return codes

Set the return code to affect the outcome of the event:

0 (Default) Accept the data value
1 Reject the data value and do not allow focus to change
2 Reject the data value but allow the focus to change

For information on setting the return code, see "About return values for DataWindow events" on page 387.

Usage

The ItemChanged event does not occur when the DataWindow control itself loses focus. If the user clicks on an Update or Close button, you will need to write a script that calls AcceptText to see if a changed value should be accepted before the button's action occurs. For information on the right way to do this, see AcceptText on page 434.

Examples

This example uses the ItemChanged event to provide additional validation; if the column is emp_name, it checks that only letters were entered in the column:

```
IF Dwo.name = "Emp_name" THEN
       IF NOT Match(Data, ^[A-za-z]$) THEN
          RETURN 2
       END IF
END IF
```

See also

ItemError

ItemError

Description

Occurs when a field has been modified, the field loses focus (for example, the user presses ENTER, TAB, or an arrow key or clicks the mouse on another field in the DataWindow), and the data in the field does not pass the validation rules for its column. ItemError can also occur when a value imported into a DataWindow control or DataStore does not pass the validation rules for its column.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

PocketBuilder event information

Event ID: pbm_dwnitemvalidationerror

Argument	Description
row	Long by value. The number of the row containing the item whose new value has failed validation.
dwo	DWObject by value. A reference to the column containing the item. Dwo is a reference to the column control, not the name of the column.
data	String by value. The new data the user specified for the item.

Return codes

Set the return code to affect the outcome of the event:

0 (Default) Reject the data value and show an error message box
1 Reject the data value with no message box
2 Accept the data value
3 Reject the data value but allow focus to change

For information on setting the return code in a particular environment, see "About return values for DataWindow events" on page 387.

Usage

If the Return code is 0 or 1 (rejecting the data), the field with the incorrect data regains the focus.

The ItemError event occurs instead of the ItemChanged event when the new data value fails a validation rule. You can force the ItemError event to occur by rejecting the value in the ItemChanged event.

Examples

The following excerpt from an ItemError event script of a DataWindow control allows the user to blank out a column and move to the next column. For columns with datatypes other than string, the user cannot leave the value empty (the empty string does not match the datatype). If the user tried to leave the value blank, this code sets the value of the column to a NULL value of the appropriate datatype.

```
string ls_colname, ls_datatype

ls_colname = dwo.Name
ls_datatype = dwo.ColType

// Reject the value if non-blank
IF Trim(data) <> "" THEN
       RETURN 0
END IF

// Set value to null if blank
CHOOSE CASE ls_datatype

       CASE "long"
       integer null_num
       SetNull(null_num)
       This.SetItem(row, ls_colname, null_num)
       RETURN 3

       CASE "date"
       date null_date
       SetNull(null_date)
       This.SetItem(row, ls_colname, null_date)
       RETURN 3

       // Additional cases for other datatypes
END CHOOSE
```

See also

ItemChanged

ItemFocusChanged

Description

Occurs when the current item in the control changes.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

PocketBuilder event information

Event ID: pbm_dwnitemchangefocus

Argument	Description
row	Long by value. The number of the row containing the item that just gained focus.
dwo	DWObject by value. A reference to the column containing the item.

Return codes

There are no special outcomes for this event. The only code is:

0 Continue processing

Usage

ItemFocusChanged occurs when focus is set to another column in the DataWindow, including when the DataWindow is first displayed. The row and column together uniquely identify an item in the DataWindow.

In the ItemFocusChanged event, dwo is always a column control. Therefore, you can get more information about it by examining any properties that are appropriate for columns such as dwo.id and dwo.Name.

Examples

This example reports the row and column that just gained focus and that just lost focus. (The first time the event occurs, there is no item that just lost focus; the script saves the row number and column name in two instance variables called ii_row and is_colname so that the old item is known the next time the event occurs.)

```
IF ii_row > 0 THEN
       sle_olditem.Text = "Old row: " + String(ii_row) &
       + " Old column: " + is_colname
END IF

sle_newitem.Text = "New row: " + String(row) &
       + " New column: " + dwo.Name

// Replace values of instance variables
// with info for next change in focus
ii_row = row
is_colname = dwo.Name
```

See also

RowFocusChanged
RowFocusChanging

KeyDown

Description

Occurs for each keystroke when the user is editing in the DataWindow edit control.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✓

PocketBuilder event information
Event ID: pbm_dwnkey

KeyDown is not a standard DataWindow event. To write a script for this event, you must first define a user-defined event for the event ID pbm_dwnkey.

Argument	Description
key	Integer by value.
keyflags	UnsignedLong by value. The modifier keys that are pressed. The keyflags value is the sum of the values for all the pressed keys. Key values are: • 1 Shift key • 2 Ctrl key • 3 Shift + Ctrl keys

Return codes

There are no special outcomes for this event. The only code is:

0 Continue processing

LoseFocus

Description

Occurs just before a control loses focus (after it becomes inactive).

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

PocketBuilder event information
Event ID: pbm_dwnkillfocus

Return codes

There are no special outcomes for this event. The only code is:

0 Continue processing

Usage

Write code for a control's LoseFocus event if you want some processing to occur when the user changes focus to another control.

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.

When to call AcceptText You should not call AcceptText in the LoseFocus event or from a user event posted from LoseFocus, unless the DataWindow control no longer has focus. For information about the right way to call AcceptText when the DataWindow control loses focus, see the AcceptText method.

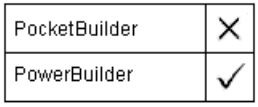
See also

GetFocus
AcceptText method

MessageText

Description

Occurs when a crosstab DataWindow generates a message. Typical messages are Retrieving data and Building crosstab.



PowerBuilder event information
Event ID: pbm_dwnmessageText

Return codes

There are no special outcomes for this event. The only code is:

0 Continue processing

MouseMove

Description

Occurs when the user moves the mouse pointer in a DataWindow control.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✓

PocketBuilder event information
Event ID: pbm_dwnmousemove

MouseMove is not a standard PocketBuilder DataWindow event. To write a script for this event, you must first define a user event for the event ID pbm_dwnmousemove.

Argument	Description
xpos	Integer by value. The distance of the pointer from the left side of the DataWindow's workspace. The distance is given in pixels.
ypos	Integer by value. The distance of the pointer from the top of the DataWindow's workspace. The distance is given in pixels.
row	Long by value. The number of the row under the pointer. If the pointer is not over a row, the value of the row argument is 0. For example, row is 0 when the user double-clicks outside the data area, in text or spaces between rows, or in the header, summary, or footer area.
dwo	DWObject by value. A reference to the control within the DataWindow that is under the pointer.

Return codes

There are no special outcomes for this event. The only code is:

0 Continue processing

Usage

The dwo, Name, or DWObject argument provides easy access to the control the user clicks. You do not need to know the coordinates of elements within the DataWindow to program control-specific responses to the user's clicks. For example, you can prevent editing of a column and use the Clicked event to set data or properties for the column and row the user clicks.

The xpos and ypos arguments provide the same values the functions PointerX and PointerY return when you call them for the DataWindow control.

See also

Clicked
DoubleClicked
MouseUp
RButtonDown

MouseUp

Description

Occurs when the user releases a mouse button in a DataWindow control.

PocketBuilder	✗
PowerBuilder	✓

PowerBuilder event information
Event ID: pbm_dwnlbuttonup

Return codes

There are no special outcomes for this event. The only code is:

0 Continue processing

PrintEnd

Description

Occurs when the printing of a DataWindow or DataStore ends.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✓

PocketBuilder event information
Event ID: pbm_dwnprintend

Argument	Description
pagesprinted	Long by value. The total number of pages that were printed.

Return codes

There are no special outcomes for this event. The only code is:

0 Continue processing

Examples

This statement displays the number of pages that were printed after the Print function was called to print the contents of the DataWindow control:

```
st_1.Text = String(pagesprinted) &
       + " page(s) have been printed."
```

See also

PrintMarginChange
PrintPage
PrintStart

PrintMarginChange

Description

Occurs when the print margins of the DataWindow change.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✓

PocketBuilder event information

Event ID: pbm_dwnprintmarginchange

PrintMarginChange is not a standard PocketBuilder DataWindow event. To write a script for this event, you must first define a user-defined event for the event ID pbm_dwnprintmarginchange.

Return codes

There are no special outcomes for this event. The only code is:

0 Continue processing

See also

PrintEnd
PrintPage
PrintStart

PrintPage

Description

Occurs before each page of the DataWindow or DataStore is formatted for printing.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✓

PocketBuilder event information

Event ID: pbm_dwnprintpage

Argument	Description
pagenumber	Long by value. The number of the page about to be printed.
copy	Long by value. The number of the copy being printed.

Return codes

Set the return code to affect the outcome of the event:

0 Do not skip the page
1 Skip the page

For information on setting the return code in a particular environment, see "About return values for DataWindow events" on page 387.

Examples

Example 1 After a script prints a DataWindow control, you can limit the number of pages to be printed by suppressing every page after page 50.

This statement in a CommandButton's Clicked event script prints the contents of the DataWindow control:

```
dw_1.Print()
```

This code in the PrintPage event of dw_1 cancels printing after reaching page 50:

```
IF pagenumber > 50 THEN This.PrintCancel()
```

Example 2 If you know every fifth page of the DataWindow contains the summary information you want, you can suppress the other pages with some arithmetic and a RETURN statement:

```
IF Mod(pagenumber / 5) = 0 THEN
       RETURN 0
ELSE
       RETURN 1
END IF
```

See also

PrintEnd
PrintMarginChange
PrintStart

PrintStart

Description

Occurs when the printing of the DataWindow or DataStore starts.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✓

PocketBuilder event information
Event ID: pbm_dwnprintstart

Argument	Description
pagesmax	Long by value. The total number of pages that will be printed, unless pages are skipped.

Return codes

There are no special outcomes for this event. The only code is:

0 Continue processing

Usage

To skip printing some of the pages in the DataWindow or DataStore, write code for the PrintPage event.

See also

PrintEnd
PrintMarginChange
PrintPage

ProcessEnter

Description

Occurs when the user presses the Enter key when focus is in the DataWindow or the DataWindow's edit control.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

PocketBuilder event information
Event ID: pbm_dwnprocessenter

ProcessEnter is not a standard PocketBuilder DataWindow event. To write a script for this event, you must first define a user-defined event for the event ID pbm_dwnprocessenter.

Return codes

There are no special outcomes for this event. The only code is:

0 Continue processing

RButtonDown

Description

Occurs when the right mouse button is pressed on the DataWindow control.

PocketBuilder	✗
PowerBuilder	✓

PowerBuilder event information
Event ID: pbm_dwnrbuttondown

Return codes

There are no special outcomes for this event. The only code is:

0 Continue processing

Resize

Description

Occurs when the user or a script opens or resizes the client area of a DataWindow control.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

PocketBuilder event information

Event ID: pbm_dwnresize

Argument	Description
sizetype	UnsignedLong by value. • 0 — (SIZE_RESTORED) The DataWindow has been resized, but it was not minimized or maximized. The user may have dragged the borders or a script may have called the Resize function. • 1 — (SIZE_MINIMIZED) The DataWindow has been minimized. • 2 — (SIZE_MAXIMIZED) The DataWindow has been maximized.
newwidth	Integer by value. The width of the client area of the DataWindow control in pixels.
newheight	Integer by value. The height of the client area of the DataWindow control in pixels.

Return codes

There are no special outcomes for this event. The only code is:

0 Continue processing

RetrieveEnd

Description

Occurs when the retrieval for the DataWindow or DataStore is complete.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

PocketBuilder event information

Event ID: pbm_dwnretrieveend

Argument	Description
rowcount	Long by value. The number of rows that were retrieved.

Return codes

There are no special outcomes for this event. The only code is:

0 Continue processing

Examples

This MessageBox displayed in the RetrieveEnd event script tells the user the number of rows just retrieved:

```
MessageBox("Total rows retrieved", String(rowcount))
```

See also

RetrieveRow
RetrieveStart
SQLPreview
UpdateStart

RetrieveRow

Description

Occurs after a row has been retrieved.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

PocketBuilder event information

Event ID: pbm_dwnretrieverow

Argument	Description
row	Long by value. The number of the row that was just retrieved

Return codes

Set the return code to affect the outcome of the event:

0 Continue processing
1 Stop the retrieval

For information on setting the return code in a particular environment, see "About return values for DataWindow events" on page 387.

Usage

If you want to guard against potentially large queries, you can have code in the RetrieveRow event check the row argument and decide whether the user has reached a maximum limit. When row exceeds the limit, you can return 1 to abort the retrieval (in which case the retrieval cannot be resumed).

A script in the RetrieveRow event (even a comment) can significantly increase the time it takes to complete a query.

Examples

This code for the RetrieveRow event aborts the retrieval after 250 rows have been retrieved.

```
IF ll_row > 250 THEN
       MessageBox("Retrieval Halted", &
           "You have retrieved 250 rows, the allowed &
           maximum.")
       RETURN 1
ELSE
       RETURN 0
END IF
```

See also

RetrieveEnd
RetrieveStart
SQLPreview
UpdateStart

RetrieveStart

Description

Occurs when the retrieval for the DataWindow or DataStore is about to begin.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

PocketBuilder event information
Event ID: pbm_dwnretrievestart

Return codes

Set the return code to affect the outcome of the event:

0 Continue processing
1 Do not perform the retrieval
2 Do not reset the rows and buffers before retrieving data

For information on setting the return code in a particular environment, see "About return values for DataWindow events" on page 387.

Usage

A return code of 2 prevents previously retrieved data from being cleared, allowing the current retrieval process to append new rows to the old data.

Examples

Example 1 This statement in the RetrieveStart event prevents a reset from taking place (rows will be added to the end of the previously retrieved rows):

```
RETURN 2
```

Example 2 This statement in the RetrieveStart event aborts the retrieval:

```
RETURN 1
```

Example 3 This code allows rows to be retrieved only when a user has an ID between 101 and 200 inclusive (the ID was stored in the instance variable il_id_number when the user started the application); all other IDs cannot retrieve rows:

```
CHOOSE CASE il_id_number
       CASE IS < 100
       RETURN 1

       CASE 101 to 200
       RETURN 0

       CASE IS > 200
       RETURN 1
END CHOOSE
```

See also

RetrieveEnd
RetrieveRow
SQLPreview
UpdateStart

RowFocusChanged

Description

Occurs when the current row changes in the DataWindow.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

PocketBuilder event information

Event ID: pbm_dwnrowchange

Argument	Description
currentrow	Long by value. The number of the row that has just become current.

Return codes

There are no special outcomes for this event. The only code is:

0 Continue processing

Usage

The SetRow function, as well as user actions, can trigger the RowFocusChanged and ItemFocusChanged events.

In a read-only DataWindow, when you click on any column that is not in the current row, the RowFocusChanging and RowFocusChanged events fire, but the current column is not changed—the current column remains at 0, since no column can have focus. DataWindows are read-only if updates are not allowed, all tab orders are set to 0, or all tab columns are protected.

If, however, focus is on an editable column in an updatable DataWindow (a DataWindow that has one or more editable columns), the row focus events do not fire when you click on a protected column or on a column whose tab order is 0. The focus remains on the current, editable column.

If focus moves off an editable column in an updatable DataWindow, the DataWindow switches to read-only mode. This can happen when the last row in the DataWindow does not have an editable column. In this case, tabbing off the last editable column causes the row focus to move to the row following the row with the last editable column. The DataWindow then remains in read-only mode until focus is given to an editable column.

Examples

This example displays the current row number and the total number of rows in a SingleLineEdit:

```
sle_1.Text = "Row " + String(currentrow) &
       + " of " + String(This.RowCount())
```

See also

ItemFocusChanged
RowFocusChanging

RowFocusChanging

Description

Occurs when the current row is about to change in the DataWindow. (The current row of the DataWindow is not necessarily the same as the current row in the database.)

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

The RowFocusChanging event occurs just before the RowFocusChanged event.

PocketBuilder event information
Event ID: pbm_dwnrowchanging

Argument	Description
currentrow	Long by value. The number of the row that is current (before the row is deleted or its number changes). If the DataWindow object is empty, currentrow is 0 to indicate there is no current row.
newrow	Long by value. The number of the row that is about to become current. If the new row is going to be an inserted row, newrow is 0 to indicate that it does not yet exist.

Return codes

Set the return code to affect the outcome of the event:

0 Continue processing (setting the current row)
1 Prevent the current row from changing

For information on setting the return code in a particular environment, see “About return values for DataWindow events” on page 387.

Usage

Typically the RowFocusChanging event is coded to respond to a mouse click or keyboard action that would change the current row in the DataWindow object. The following functions can also trigger the RowFocusChanging event, as well as the RowFocusChanged and ItemFocusChanged events, when the action changes the current row:

SetRow
Retrieve

RowsCopy
RowsMove
DeleteRow
RowsDiscard

In these cases, the RowFocusChanging event script can prevent the changing of the DataWindow object's current row only. The script cannot prevent the data from being changed (for example, the rows still get moved).

If you set the RowFocusChanging return value to 1 for a freeform DataWindow, the current row does not change, but the DataWindow still scrolls in response to a ScrollToRow function call.

In a tabular DataWindow, if the user clicks to change rows, the row focus does not change, and the row and DataWindow do not scroll. You can still scroll programmatically or by using the scroll bar.

In a read-only DataWindow, when you click on any column that is not in the current row, the RowFocusChanging and RowFocusChanged events fire, but the current column is not changed—the current column remains at 0, since no column can have focus. DataWindows are read-only if updates are not allowed, all tab orders are set to 0, or all tab columns are protected.

However, if focus is on an editable column in an updatable DataWindow (a DataWindow that has one or more editable columns), the row focus events do not fire when you click on a protected column or on a column whose tab order is 0. The focus remains on the current, editable column.

If focus moves off an editable column in an updatable DataWindow, the DataWindow switches to read-only mode. This can happen when the last row in the DataWindow does not have an editable column. In this case, tabbing off the last editable column causes the row focus to move to the row following the row with the last editable column. The DataWindow then remains in read-only mode until focus is given to an editable column.

Examples

This example displays a message alerting you that changes have been made in the window dw_detail that will be lost if the row focus is changed to the window dw_master.

```
IF dw_detail.DeletedCount() > 0 OR &
          dw_detail.ModifiedCount() > 0 THEN
       MessageBox("dw_master", "Changes will be lost &
          in Detail")
ELSE
       RETURN 0
END IF
```

See also

ItemFocusChanged
RowFocusChanged

ScrollHorizontal

Description

Occurs when user scrolls left or right in the DataWindow with the TAB or arrow keys or the scroll bar.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

PocketBuilder event information

Event ID: pbm_dwnhscroll

Argument	Description
scrollpos	Long by value. The distance in PowerBuilder units of the scroll box from the left end of the scroll bar (if the DataWindow is split, in the pane being scrolled).
pane	Integer by value. The number of the pane being scrolled. (When the DataWindow is split with two scroll bars, there are two panes.) Values are: • 1 — The left pane (if the scroll bar is not split, the only pane). • 2 — The right pane.

Return codes

There are no special outcomes for this event. The only code is:

0 Continue processing

Examples

This example displays the customer ID of the current row (the cust_id column) in a SingleLineEdit control when the pane being scrolled is pane 1 and the position is greater than 100:

```
string ls_id
ls_id = ""
IF pane = 1 THEN
       IF scrollpos > 100 THEN
          ls_id =
String(dw_1.Object.Id[dw_1.GetRow()])
       END If
END IF
```

```
sle_message.Text = ls_id

RETURN 0
```

See also ScrollVertical

ScrollVertical

Description Occurs when user scrolls up or down in the DataWindow with the TAB or arrow keys or the scroll bar.

PocketBuilder	✗
PowerBuilder	✓

PowerBuilder event information
Event ID: pbm_dwnvscroll

Return codes There are no special outcomes for this event. The only code is:

0 Continue processing

See also ScrollHorizontal

SQLPreview

Description Occurs immediately before a SQL statement is submitted to the DBMS. Functions that trigger DBMS activity are Retrieve, Update, and ReselectRow.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

PocketBuilder event information
Event ID: pbm_dwnsql

Argument	Description
request	SQLPreviewFunction by value. The function that initiated the database activity. For a list of valid values, see SQLPreviewFunction on page 379.

Argument	Description
sqltype	SQLPreviewType by value. The type of SQL statement being sent to the DBMS. For a list of valid values, see SQLPreviewType on page 379.
sqlsyntax	String by value. The full text of the SQL statement.
buffer	DWBuffer by value. The buffer containing the row involved in the database activity. For a list of valid values, see DWBuffer on page 372.
row	Long by value. The number of the row involved in the database activity, that is, the row being updated, selected, inserted, or deleted.

Return codes

Set the return code to affect the outcome of the event:

0 Continue processing
1 Stop processing
2 Skip this request and execute the next request

For information on setting the return code in a particular environment, see "About return values for DataWindow events" on page 387.

Usage

Some uses for the sqlsyntax argument are:

- Changing the SQL to be executed (you can get the value of sqlsyntax, modify it, and call SetSQLPreview)
- Keeping a record (you can write the SQL statement to a log file)

Reported row number
The row number stored in row is the number of the row in the buffer, not the number the row had when it was retrieved into the DataWindow object.

If the row that caused the error is in the Filter buffer, you must unfilter it if you want the user to correct the problem.

GetSQLPreview and binding
When binding is enabled for your database, the SQL returned in the GetSQLPreview event may not be complete—the input arguments are not replaced with the actual values. For example, when binding is enabled, GetSQLPreview might return the following SQL statement:

```
INSERT INTO "cust_order" ( "ordnum", "custnum",
"duedate", "balance" ) VALUES ( ?, ?, ?, ? )
```

When binding is disabled, it returns:

```
INSERT INTO "cust_order" ( "ordnum", "balance",
"duedate", "custnum" ) VALUES ( '12345', 900,
'3/1/94', '111' )
```

If you require the complete SQL statement for logging purposes, you should disable binding in your DBMS.

Examples

This statement in the SQLPreview event sets the current SQL string for the DataWindow dw_1:

```
dw_1.SetSQLPreview( &
       "INSERT INTO billings VALUES(100, " + &
       String(Current_balance) + ")")
```

See also

RetrieveStart
UpdateEnd
UpdateStart

TabDownOut

Description

Occurs when the user presses Enter or the down arrow to change focus to the next item in a DataWindow column.

PocketBuilder	✗
PowerBuilder	✓

PowerBuilder event information
Event ID: pbm_dwntabdownout

Return codes

There are no special outcomes for this event. The only code is:

0 Continue processing

TabOut

Description

Occurs when the user presses Tab or, in some edit styles, the right arrow, to move to the next cell in the DataWindow.

PocketBuilder	✗
PowerBuilder	✓

PowerBuilder event information
Event ID: pbm_dwntabout

Return codes

There are no special outcomes for this event. The only code is:

0 Continue processing

TabUpOut

Description

Occurs when the user presses Shift+Enter or the up arrow to move to the previous item in a DataWindow column.

PocketBuilder	✗
PowerBuilder	✓

PowerBuilder event information
Event ID: pbm_dwntabupout

Return codes

There are no special outcomes for this event. The only code is:

0 Continue processing

UpdateEnd

Description

Occurs when all the updates to the database from the DataWindow (or DataStore) are complete.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

PocketBuilder event information
Event ID: pbm_dwnupdateend

Argument	Description
rowsinserted	Long by value. The number of rows inserted.
rowsupdated	Long by value. The number of rows updated.
rowsdeleted	Long by value. The number of rows deleted.

Return codes

There are no special outcomes for this event. The only code is:

0 Continue processing

See also

RetrieveStart
SQLPreview
UpdateStart

UpdateStart

Description

Occurs after a script calls the Update function and just before changes in the DataWindow or DataStore are sent to the database.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

PocketBuilder event information
Event ID: pbm_dwnupdatestart

Return codes

Set the return code to affect the outcome of the event:

0 Continue processing
1 Do not perform the update

For information on setting the return code in a particular environment, see “About return values for DataWindow events” on page 387.

See also

RetrieveStart
SQLPreview
UpdateEnd

CHAPTER 9

Methods for the DataWindow Control

About this chapter

This chapter documents the methods of the DataWindow control, providing method syntax, notes, and examples.

Methods for graphs are in Chapter 10, “Methods for Graphs in the DataWindow Control.”

Contents

The methods in this chapter are listed alphabetically.

Before you begin

For methods (or functions) that apply to controls other than DataWindows and DataStores, see the *PowerScript Reference*.

AboutBox

Description

Displays a dialog identifying the DataWindow, including copyright and version information.

PocketBuilder	✗
PowerBuilder	✓

Syntax

Web ActiveX DataWindow control

void *dwcontrol*.**AboutBox** ( )

Return value

None

AcceptText

Description

Applies the contents of the DataWindow's edit control to the current item in the buffer of a DataWindow control or DataStore. The data in the edit control must pass the validation rule for the column before it can be stored in the item.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

integer *dwcontrol*.**AcceptText** ( )

Argument	Description
dwcontrol	A reference to a DataWindow control, DataStore, or child DataWindow

Return value

Returns 1 if it succeeds and -1 if it fails (for example, the data did not pass validation).

If *dwcontrol* is NULL, the method returns NULL. If there is no DataWindow object assigned to the DataWindow control or DataStore, this method returns 1.

Usage

When a user moves from item to item in a DataWindow control, the control validates and accepts data the user has edited.

How to call AcceptText When a user modifies a DataWindow item and then immediately changes focus to another control in the window, the DataWindow control does not accept the modified data—the data remains in the edit control. Use the AcceptText method in this situation to ensure that the DataWindow object contains the data the user edited.

However, you must not call AcceptText in the LoseFocus event or in a user event posted from LoseFocus if the DataWindow control still has focus. If you do, an infinite loop can occur.

The problem Normally, new data is validated and accepted when the user moves to a new cell in the DataWindow. If the new data causes an error, a message box displays, which causes the DataWindow to lose focus. If you have also coded the LoseFocus event or an event posted from LoseFocus to call AcceptText to validate data when the control loses focus, this AcceptText runs because of the message box and triggers an infinite loop of validation errors.

The solution It is desirable to validate the last changed data when the control loses focus. You can accomplish this by making sure AcceptText gets called only when the DataWindow control really has lost focus. The third example below illustrates how to use an instance variable to keep track of whether the DataWindow control has focus. The posted event calls AcceptText only when the DataWindow control does not have focus.

Events AcceptText can trigger an ItemChanged or an ItemError event.

AcceptText in the ItemChanged event
Calling AcceptText in the ItemChanged event has no effect.

Examples

Example 1 In this example, the user is expected to enter a key value (such as an employee number) in a column of the DataWindow object, then click the OK button. This script for the Clicked event for the button calls AcceptText to validate the entry and place it in the DataWindow control. Then the script uses the item in the Retrieve method to retrieve the row for that key:

```
IF dw_emp.AcceptText() = 1 THEN
       dw_emp.Retrieve(dw_emp.GetItemNumber &
           (dw_emp.GetRow(), dw_emp.GetColumn()))
END IF
```

Example 2 This script for the Clicked event for a CommandButton accepts the text in the DataWindow dw_Emp and counts the rows in which the column named balance is greater than 0:

```
integer i, Count
dw_employee.AcceptText()
FOR i = 1 to dw_employee.RowCount()
       IF dw_employee.GetItemNumber(i,'balance') &
              > 0 THEN
          Count = Count + 1
       END IF
NEXT
```

Example 3 This example illustrates how to validate newly entered data when the DataWindow control loses focus. An instance variable keeps track of whether the DataWindow control has focus. It is set in the GetFocus and LoseFocus events. The LoseFocus event posts the ue_acceptText event, which calls the AcceptText method only if the DataWindow control does not have focus.

The instance variable:

```
boolean dw_has_focus
```

The GetFocus event:

```
dw_has_focus = TRUE
```

The LoseFocus event:

```
dw_has_focus = FALSE
dw_1.event  post ue_acceptText( )
```

The ue_acceptText event:

```
IF dw_has_focus = FALSE THEN
       dw_1.accepttext( )
END IF
```

See also Update

CanUndo

Description Tests whether Undo can reverse the most recent edit in the editable control over the current row and column.

PocketBuilder	✗
PowerBuilder	✓

Syntax **PowerBuilder**

boolean *dwcontrol*.**CanUndo** ( )

Argument	Description
dwcontrol	A reference to a DataWindow control

Return value

Returns TRUE if the last edit can be reversed (undone) using the Undo method and FALSE if the last edit cannot be reversed.

If *dwcontrol* is NULL, the method returns NULL.

ClassName

Description

Provides the class (or name) of the specified object.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

string *dwcontrol*.**ClassName** ( )

Argument	Description
dwcontrol	A reference to a DataWindow control

Return value

Returns the class of *dwcontrol*, the name assigned to the control. Returns the empty string ("") if an error occurs.

If *dwcontrol* is NULL, the method returns NULL.

Usage

Method inherited from PowerObject. For use with variables in the PocketBuilder environment, see ClassName in *PowerScript Reference*.

Clear

Description

Deletes selected text in the edit control over the current row and column, but does not store it in the clipboard.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

long *dwcontrol*.**Clear** ( )

Argument	Description
dwcontrol	A reference to a DataWindow control

Return value

Returns the number of characters that Clear removed from *dwcontrol*. If no text is selected, no characters are removed and Clear returns 0. If an error occurs, Clear returns -1.

If *dwcontrol* is NULL, the method returns NULL.

Usage

To select text for deleting, the user can use the mouse or keyboard. You can also call the SelectText method in a script. To delete selected text and store it in the clipboard, use the Cut method.

Using with other controls
For use with other PocketBuilder controls, see Clear in the *PowerScript Reference*.

Examples

If the user is editing the emp_name column in dw_emp and selects the text Wilson, this statement clears Wilson from the edit control and returns 6:

```
long chars_returned
chars_returned = dw_emp.Clear( )
```

If the text in the edit control in dw_emp is Wilson, the first statement selects the W and the second clears W from the edit control. The return value would be 1:

```
dw_emp.SelectText(1,1)
dw_emp.Clear( )
```

See also

Clear in the *PowerScript Reference*
Cut
Paste
ReplaceText
SelectText

ClearValues

Description

Deletes all the items from a value list or code table associated with a DataWindow column. (A value list is called a code table when it has both display and data values.) ClearValues does not affect the data stored in the column.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

integer *dwcontrol*.**ClearValues** (string *column*)

integer *dwcontrol*.**ClearValues** (integer *column*)

Argument	Description
dwcontrol	A reference to a DataWindow control, DataStore, or child DataWindow.
column	The column whose value list you want to delete. *Column* can be a column number (integer) or a column name (string).

Return value

Returns 1 if it succeeds and -1 if an error occurs. The return value is usually not used.

Usage

The edit style of the column can be DropDownListBox, Edit, or RadioButton. ClearValues has no effect when *column* has the EditMask or DropDownDataWindow edit style.

Examples

This statement clears all values from the drop-down list of dw_Employee's status column:

```
dw_Employee.ClearValues("status")
```

See also

GetValue
SetValue

Copy

Description

Puts selected text from the current row and column of an edit control onto the clipboard. Copy does not change the source text.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

integer *objectref*.**Copy** ( )

Argument	Description
objectref	A reference to a DataWindow control *or* (PowerBuilder only) 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*

Return value

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.

For OLE DWObjects, Copy returns 0 if it succeeds and one of the following negative values if an error occurs:

-1 Container is empty
-2 Copy Failed
-9 Other error

If *objectref* is NULL, the method returns NULL.

Usage

To select text for copying, the user can use the mouse or keyboard. You can also call the SelectText method in a script.

To insert the contents of the clipboard into a control, use the Paste method.

Copy does not delete the selected text or OLE object. To delete the data, use the Clear or Cut method.

Using with other controls
For use with other PocketBuilder controls, see Copy in the *PowerScript Reference*.

Examples

Assuming the selected text in the edit control of dw_emp is Temporary Address, these statements copy Temporary Address to the clipboard and store 17 in copy_amt:

```
integer copy_amt
copy_amt = dw_emp.Copy()
```

See also

Clear
Clipboard in the *PowerScript Reference*
Cut
Paste
ReplaceText
SelectText

CopyRTF

Description

Returns the selected text, pictures, and input fields in a RichText DataWindow as a string with rich text formatting. Bitmaps and input fields are included in the string.

PocketBuilder	✗
PowerBuilder	✓

Syntax

PowerBuilder

string *dwcontrol*.**CopyRTF** ({ boolean *selected* {, Band *band* } })

Return value

Returns the selected text as a string.

CopyRTF returns an empty string ("") if:

- There is no selection and *selected* is TRUE
- An error occurs

Create

Description

Creates a DataWindow object using DataWindow source code and puts that object in the specified DataWindow control or DataStore object. This dynamic DataWindow object does not become a permanent part of the application source library.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

integer *dwcontrol*.**Create** (string *syntax* {, string *errorbuffer* })

Argument	Description
dwcontrol	A reference to the DataWindow control or DataStore in which PocketBuilder will create the new DataWindow object.
syntax	A string whose value is the DataWindow source code that will be used to create the DataWindow object.
errorbuffer (optional)	The name of a string that will hold any error messages that are generated. If you do not specify an error buffer, a message box will display the error messages.

Return value

Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is NULL, the method returns NULL.

Usage

The Create method creates a DataWindow object using the source code in *syntax*. It substitutes the new DataWindow object for the DataWindow object currently associated with *dwcontrol*.

DataWindow source code syntax is complex and is best produced by copying existing DataWindows. In the PocketBuilder development environment, you can export the syntax of a DataWindow object in the Library painter. In a PowerBuilder application, you can use the Describe and LibraryExport methods to obtain the source code of existing DataWindows to use as models. The LibraryExport method is not supported in PocketBuilder.

Another source of DataWindow code is the SyntaxFromSQL function, which creates DataWindow source code based on a SQL statement. Many values in the source code syntax correspond to properties of the DataWindow object, which are documented in Chapter 3, "DataWindow Object Properties."

When you examine syntax for existing DataWindow objects, you will see that the order of the syntax can vary. Release must be the first statement, and DataWindow should be the next statement. If you change the order, use care; the order can affect the results.

Calling SyntaxFromSQL as the syntax argument
You can call SyntaxFromSQL directly as the value for *syntax*. However, this does not give you the chance to check whether errors have been reported in its error argument. Before you use SyntaxFromSQL in Create, make sure the SQL syntax is valid.

Comments To designate text in your DataWindow syntax as a comment, use either of the following standard comment indicators:

- Use double slashes (//) to indicate that the text after the slashes and on the same line is a comment.

When you use this method, the comment can be all or part of a line but cannot cover multiple lines; the compiler ignores everything following the double slashes on the line.

- Begin a comment with slash asterisk (/*) and end it with asterisk slash (*/) to indicate that all the text between the delimiters is a comment.

  When you use this method, the comment can be all or part of a line or occupy multiple lines; the compiler ignores everything between /* and */.

Examples

These statements create a new DataWindow in the control dw_new from the DataWindow source code returned by the SyntaxFromSQL function. Errors from SyntaxFromSQL and Create are displayed in the MultiLineEdits mle_sfs and mle_create. After creating the DataWindow, you must call SetTransObject for the new DataWindow object before you can retrieve data:

```
string error_syntaxfromSQL, error_create
string new_sql, new_syntax

new_sql = 'SELECT emp_data.emp_id, ' &
       + 'emp_data.emp_name ' &
       + 'from emp_data ' &
       + 'WHERE emp_data.emp_salary>45000'

new_syntax = SQLCA.SyntaxFromSQL(new_sql, &
       'Style(Type=Form)', error_syntaxfromSQL)

IF Len(error_syntaxfromSQL) > 0 THEN
       // Display errors
       mle_sfs.Text = error_syntaxfromSQL
ELSE
       // Generate new DataWindow
       dw_new.Create(new_syntax, error_create)
       IF Len(error_create) > 0 THEN
          mle_create.Text = error_create
       END IF
END IF

dw_new.SetTransObject(SQLCA)
dw_new.Retrieve()
```

See also

SyntaxFromSQL in *PowerScript Reference*
SetTrans
SetTransObject

CreateError

Description

Returns the error messages that were generated during a previous call to Create.

PocketBuilder	✗
PowerBuilder	✓

Syntax

Web ActiveX DataWindow control

string *dwcontrol*.**CreateError** ( )

Return value

Returns a string whose value is the error message text that was generated when creating a DataWindow from source code. If no errors occur, returns an empty string.

CreateFrom

Description

Creates a DataStore object from the passed ResultSet object.

PocketBuilder	✗
PowerBuilder	✓

Syntax

PowerBuilder DataStore object

integer *dsobject*.**CreateFrom** (ResultSet *rssource*)

Return value

Integer. Returns 1 if it succeeds or a negative number if an error occurs. If any argument is NULL, the method returns NULL.

CrosstabDialog

Description

Displays the Crosstab Definition dialog box so the user can modify the definition of a crosstab DataWindow during execution. The dialog box is the one you use in the DataWindow painter to define the crosstab.

PocketBuilder	✗
PowerBuilder	✓

Syntax

PowerBuilder DataWindow control

integer *dwcontrol*.**CrossTabDialog** ( )

Return value

Returns 1 if it succeeds and -1 if an error occurs.

If *dwcontrol* is NULL, the method returns NULL.

Cut

Description

Deletes selected text in the current row and column of an edit control and stores it on the clipboard, replacing the clipboard contents with the deleted text.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

long *dwcontrol*.**Cut** ( )

Argument	Description
dwcontrol	A reference to a DataWindow control. The text is cut from the edit control over the current row and column.

Return value

Returns the number of characters that were cut from *dwcontrol* 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 *dwcontrol* is NULL, the method returns NULL.

Usage

To select text for deleting, the user can use the mouse or keyboard. You can also call the SelectText method in a script. (For the RichTextEdit presentation style in PowerBuilder, there are several additional methods for selecting text: SelectTextAll, SelectTextLine, and SelectTextWord.)

To insert the contents of the clipboard into a control, use the Paste method.

To delete selected text but not store it in the clipboard, use the Clear method.

Using with other controls
For use with other PocketBuilder controls, see Cut in the *PowerScript Reference*.

Examples

Assuming the selected text in the edit control of dw_emp is Temporary, this statement deletes Temporary from the edit control, stores it in the clipboard, and returns 9:

```
dw_emp.Cut()
```

See also

Copy
Clear
Clipboard in the *PowerScript Reference*
Paste

DBCancel

Description

Cancels the retrieval in process in a DataWindow.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

integer *dwcontrol*.**DBCancel** ( )

Argument	Description
dwcontrol	A reference to the DataWindow control, DataStore, or child DataWindows

Return value

Returns 1 if it succeeds and -1 if an error occurs. If *dwcontrol* is NULL, the method returns NULL.

Usage

To cancel a database retrieval, you need two pieces of code:

- Code that calls DBCancel. To let the user cancel the retrieval, you could call DBCancel (or call a user function or member method that calls it) in code for a button or an item on a menu. This code would generally set an instance variable or data member to indicate that the user requested cancellation:

```
ib_cancel = TRUE
dw_1.DBCancel()
```

- Code for the RetrieveRow event that sets an action/return code of 1 to stop the retrieval:

```
IF ib_cancel = TRUE THEN
       RETURN 1
END IF
```

Coding something in the RetrieveRow event's script (even just a comment) enables the operating system to process events while the DataWindow is being populated with rows from the database. If the RetrieveRow event's script is empty, menus and command buttons cannot even be clicked until the retrieval is completely finished. This can be frustrating if the user inadvertently starts a retrieval that is going to take a long time.

If the Async DBParm parameter is set to 1 (for asynchronous operation), a user or a script can cancel a query either before the first row is returned or during the data retrieval process. If Async is set to 0 (for synchronous operation), the user cannot select the menu or CommandButton until the first row is retrieved. The asynchronous setting is useful when a query might take a long time to retrieve its first row.

Examples

PowerBuilder In this example, the menu bar for an MDI application has menu items for starting and canceling a retrieval. When the user cancels the retrieval, a user function calls DBCancel and sets a boolean instance variable to TRUE. The RetrieveStart and RetrieveRow events check this variable and return the appropriate value.

In this hypothetical application, the user starts a retrieval by selecting Retrieve from a menu. The script for the Retrieve menu item calls a user function for the window:

```
w_async1.wf_retrieve()
```

The wf_retrieve function sets the Async DBParm for asynchronous processing and starts the retrieval. Because Async is set to 1, the user can select the Cancel menu item at any time, even before the first row is retrieved. (In your own application, you would include error handling to make sure Retrieve returned successfully.)

```
long rc
ib_cancel = FALSE
SQLCA.DBParm = 'Async = 1'
rc = dw_1.Retrieve()
```

The user can stop the retrieval by selecting Cancel from the menu. The script for the Cancel menu item reads:

```
w_async1.wf_cancel()
```

The user function wf_cancel for the window w_async1 calls DBCancel and sets a flag indicating that the retrieval is canceled. Other events for the DataWindow will check this flag and abort the retrieval too. The variable ib_cancel is an instance variable for the window:

```
ib_cancel = TRUE
```

```
dw_1.DBCancel()
```

Scripts for the RetrieveStart and RetrieveRow events both check the ib_cancel instance variable and, if it is TRUE, stop the retrieval by returning a value of 1. In order to cancel the retrieval, some code or comment in the script for the RetrieveRow event is required:

```
IF ib_cancel = TRUE THEN
       RETURN 1
END IF
```

See also

Retrieve

DBErrorCode

Description

Reports the database-specific error code that triggered the DBError event.

PocketBuilder	✗
PowerBuilder	✓

Syntax

PowerBuilder

long *dwcontrol*.**DBErrorCode** ( )

Argument	Description
dwcontrol	A reference to a DataWindow control or child DataWindow

Return value

Returns an error code when a database error occurs in *dwcontrol*. Error codes -1 through -4 are PowerBuilder codes. Other codes are database-specific. Returns 0 if there is no error. If *dwcontrol* is NULL, the method returns NULL.

DBErrorMessage

Description

Reports the database-specific error message that triggered the DBError event.

PocketBuilder	✗
PowerBuilder	✓

Syntax

PowerBuilder

string *dwcontrol*.**DBErrorMessage** ( )

Return value

Returns a string whose value is a database-specific error message generated by a database error in *dwcontrol*. Returns the empty string ("") if there is no error. If *dwcontrol* is NULL, the method returns NULL.

DeletedCount

Description

Reports the number of rows that have been marked for deletion in the database.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

long *dwcontrol*.**DeletedCount** ( )

Argument	Description
dwcontrol	A reference to a DataWindow control, DataStore, or child DataWindow

Return value

Returns the number of rows that have been deleted from *dwcontrol* but not updated in the associated database table. Returns 0 if no rows have been deleted or if all the deleted rows have been updated in the database table. DeletedCount returns -1 if it fails. If any argument's value is NULL, the method returns NULL.

Usage

An updatable DataWindow control or DataStore has several buffers. The primary buffer stores the rows currently being displayed. The delete buffer stores rows that the application has marked for deletion by calling the DeleteRow method. These rows are saved until the database is updated. You can use DeletedCount to find out if there are any rows in the delete buffer.

If a DataWindow is not updatable, rows that are deleted are discarded—they are not stored in the delete buffer. Therefore, DeletedCount returns 0 for a nonupdatable DataWindow unless a method, such as RowsCopy or RowsMove, has been used to populate the delete buffer.

Examples

Assuming two rows in dw_employee have been deleted but have not been updated in the associated database table, these statements set ll_Del to 2:

```
Long ll_Del
ll_Del = dw_employee.DeletedCount( )
```

This example tests whether there are rows in the delete buffer, and if so, updates the database table associated with dw_employee:

```
Long ll_Del
ll_Del = dw_employee.DeletedCount()
IF ll_Del <> 0 THEN dw_employee.Update()
```

See also

DeleteRow
FilteredCount
ModifiedCount
RowCount

DeleteRow

Description

Deletes a row from a DataWindow control, DataStore object, or child DataWindow.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

integer *dwcontrol*.**DeleteRow** (long *row*)

Argument	Description
dwcontrol	A reference to a DataWindow control, DataStore, or child DataWindow.
row	A value identifying the row you want to delete. To delete the current row, specify 0 for *row*.

Return value

Returns 1 if the row is successfully deleted and -1 if an error occurs. If any argument's value is NULL, the method returns NULL. If there is no DataWindow object assigned to the DataWindow control or DataStore, this method returns -1.

Usage

DeleteRow deletes the row from the DataWindow's primary buffer.

If the DataWindow is not updatable, all storage associated with the row is cleared. If the DataWindow is updatable, DeleteRow moves the row to the DataWindow's delete buffer; PocketBuilder uses the values in the delete buffer to build the SQL DELETE statement.

The row is not deleted from the database table until the application calls the Update method. After the Update method has updated the database and the update flags are reset, the storage associated with the row is cleared.

Examples

This statement deletes the current row from dw_employee:

```
dw_employee.DeleteRow(0)
```

These statements delete row 5 from dw_employee and then update the database with the change:

```
dw_employee.DeleteRow(5)
dw_employee.Update()
```

See also

DeletedCount
InsertRow

Describe

Description

Reports the values of properties of a DataWindow object and controls within the DataWindow object.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

string *dwcontrol*.**Describe** (string *propertylist*)

Argument	Description
dwcontrol	A reference to a DataWindow control, DataStore, or child DataWindow.
propertylist	A string whose value is a blank-separated list of properties or Evaluate functions. For a list of valid properties, see Chapter 3, "DataWindow Object Properties."

Return value

Returns a string that includes a value for each property or Evaluate function. A newline character (~n or \n) separates the value of each item in *propertylist*.

If the property list contains an invalid item, Describe returns an exclamation point (!) for that item and ignores the rest of the property list. Describe returns a question mark (?) if there is no value for a property.

When the value of a property contains an exclamation point or a question mark, the value is returned in quotes so that you can distinguish between the returned value and an invalid item or a property with no value.

If any argument's value is NULL, the method returns NULL.

Usage

Each column and graphic control in the DataWindow has a set of properties (listed in Chapter 3, "DataWindow Object Properties"). You specify one or more properties as a string, and Describe returns the values of the properties.

Describe can also evaluate expressions involving values of a particular row and column. When you include Describe's Evaluate function in the property list, the value of the evaluated expression is included in the reported information.

Use Describe to understand the structure of a DataWindow. For example, you can find out which bands the DataWindow uses and what the datatypes of the columns are. You can also use Describe to find out the current value of a property and use that value to make further modifications.

Describe is often used to obtain the DataWindow's SELECT statement in order to modify it (for example, by adding a WHERE clause).

When you can obtain the DataWindow's SQL statement
When you use the Select painter to graphically create a SELECT statement, PocketBuilder saves its own SELECT statement (called a PBSELECT statement) and not a SQL SELECT statement, with the DataWindow definition.

When you call Describe with the property Table.Select, it returns a SQL SELECT statement *only if* you are connected to the database. If you are not connected to the database, Describe returns a PBSELECT statement.

Property syntax The syntax for a property in the property list is:

controlname.property

For the types of controls in a DataWindow and their properties with examples, see Chapter 3, "DataWindow Object Properties."

Properties whose values are a list When a property returns a list, the tab character separates the values in the list. For example, the Bands property reports all the bands in use in the DataWindow as a list.

header[tab]detail[tab]summary[tab]footer[tab]header.1[tab]trailer.1

If the first character in a property's returned value list is a quotation mark, it means the whole list is quoted and any quotation marks within the list are single quotation marks. For example, the following is a single property value.

" Student[tab]'Andrew'or'[newline]Andy' "

Specifying special characters Table 9-1 shows how you specify special characters in a string.

Table 9-1: Specifying special characters

Character	PocketBuilder
tab	~t
newline	~n
single quote	~'
double quote	~"

Quoted property values Describe returns a property's value enclosed in quotes when the text would otherwise be ambiguous. For example, if the property's value includes a question mark, then the text is returned in quotes. A question mark without quotes means that the property has no value.

Column name or number When the control is a column, you can specify the column name or a pound sign (#) followed by the column number. For example, if salary is column 5, then "salary.coltype" is equivalent to "#5.coltype".

Control names The DataWindow painter automatically gives names to all controls.

Evaluating an expression Describe's Evaluate function allows you to evaluate DataWindow painter expressions within a script using data in the DataWindow. Evaluate has the following syntax, which you specify for *propertylist*.

Evaluate ('*expression*', *rownumber*)

Expression is the expression you want to evaluate and *rownumber* is the number of the row for which you want to evaluate the expression. The expression usually includes DataWindow painter functions. For example, in the following statement, Describe reports either 255 or 0 depending on the value of the salary column in row 3:

```
ls_ret = dw_1.Describe( &
"Evaluate('If(salary > 100000, 255, 0)', 3)")
```

You can call DataWindow control functions in a script to get data from the DataWindow, but some painter functions (such as LookUpDisplay) cannot be called in a script. Using Evaluate is the only way to call them. (See the example "Evaluating the display value of a DropDownDataWindow" on page 455.)

Sample property values To illustrate the types of values that Describe reports, consider a DataWindow called dw_emp with one group level. Its columns are named emp and empname, and its headers are named emp_h and empname_h. The following table shows several properties and the returned value. In the first example below, a sample command shows how you might specify these properties for Describe and what it reports.

Table 9-2: Examples of return values for Describe method

Property	Reported value
datawindow.Bands	header[tab]detail[tab]summary[tab]footer[tab]header.1[tab]trailer.1
datawindow.Objects	emp[tab]empname[tab]emp_h[tab]empname_h
emp.Type	column
empname.Type	column
empname_h.Type	text
emp.Coltype	char(20)
state.Type	! (! indicates an invalid item — there is no column named state)
empname_h.Visible	?

Examples

This example calls Describe with some of the properties shown in the previous table. The reported values (formatted with tabs and newlines) follow. Note that because state is not a column in the DataWindow, state.type returns an exclamation point (!):

```
string ls_request, ls_report

ls_request = "DataWindow.Bands DataWindow.Objects "&
       + "empname_h.Text " &
       + "empname_h.Type emp.Type emp.Coltype " &
       + "state.Type empname.Type empname_h.Visible"

ls_report = dw_1.Describe(ls_request)
```

Describe sets the value of ls_report to the following string:

```
header~tdetail~tsummary~tfooter~theader.1~ttrailer.1~N
emp~tempname~temp_h~tempname_h~N "Employee~R~NName"~N
text~N column~Nchar(20)~N!
```

These statements check the datatype of the column named salary before using GetItemNumber to obtain the salary value:

```
string ls_data_type
integer li_rate
```

```
ls_data_type = dw_1.Describe("salary.ColType")
IF ls_data_type = "number" THEN
li_rate = dw_1.GetItemNumber(5, "salary")
ELSE
     . . . // Some processing
END IF
```

Column name or number This statement finds out the column type of the current column, using the column name:

```
s = This.Describe(This.GetColumnName()+ ".ColType")
```

For comparison, this statement finds out the same thing, using the current column's number:

```
s = This.Describe("#" + String(This.GetColumn()) &
       + ".ColType")
```

Scrolling and the current row This example, as part of the DataWindow control's ScrollVertical event, makes the first visible row the current row as the user scrolls through the DataWindow:

```
s = This.Describe("DataWindow.FirstRowOnPage")
IF IsNumber(s) THEN This.SetRow(Integer(s))
```

Evaluating the display value of a DropDownDataWindow This example uses Describe's Evaluate function to find the display value in a DropDownDataWindow column called state_code. You must execute the code *after* the ItemChanged event, so that the value the user selected has become the item value in the buffer. This code is the script of a custom user event called getdisplayvalue:

```
string rownumber, displayvalue

rownumber = String(dw_1.GetRow())
displayvalue = dw_1.Describe( &
       "Evaluate('LookUpDisplay(state_code) ', " &
       + rownumber + ")")
```

This code, as part of the ItemChanged event's script, posts the getdisplayvalue event:

```
dw_1.PostEvent("getdisplayvalue")
```

Assigning null values based on the column's datatype The following excerpt from the ItemError event script of a DataWindow control allows the user to blank out a column and move to the next column. For columns with datatypes other than string, the user cannot leave the value empty (which is an empty string and does not match the datatype) without the return code. Data and row are arguments of the ItemError event:

```
string s
s = This.Describe(This.GetColumnName() &
       + ".Coltype")

CHOOSE CASE s
       CASE "number"
       IF Trim(data) = "" THEN
           integer null_num
           SetNull(null_num)
           This.SetItem(row, &
               This.GetColumn(), null_num)
           RETURN 3
       END IF

CASE "date"
       IF Trim(data) = "" THEN
       date null_date
       SetNull(null_date)
       This.SetItem(row, &
       This.GetColumn(), null_date)
       RETURN 3
       END IF

       . . . // Additional cases for other datatypes

END CHOOSE
```

See also

Create
Modify

Drag

Description

Starts or ends the dragging of a control.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✗
PowerBuilder	✓

Syntax

integer *dwcontrol*.**Drag** (DragMode *dragvalue*)

Argument	Description
dwcontrol	A reference to a DataWindow control or child DataWindow.
dragvalue	A value indicating the action you want to take on a control: • Begin! — Put *dwcontrol* in drag mode. • Cancel! — Stop dragging *dwcontrol* but do not cause a DragDrop event. • End! — Stop dragging *dwcontrol* and if *dwcontrol* is over a target object, cause a DragDrop event.

Return value

Returns 1 if it succeeds and -1 if an error occurs.

Usage

Inherited from DragObject. For information, see Drag in the *PowerScript Reference*.

Filter

Description

Displays rows in a DataWindow that pass the current filter criteria. Rows that do not meet the filter criteria are moved to the filter buffer.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

integer *dwcontrol*.**Filter** ( )

Argument	Description
dwcontrol	A reference to a DataWindow control, DataStore, or child DataWindow

Return value

Returns 1 if it succeeds and -1 if an error occurs. The return value is usually not used.

If *dwcontrol* is NULL, the method returns NULL.

Usage

Filter causes all rows to be retrieved and then it applies the filter. Even when the Retrieve As Needed option is set, the Filter method retrieves all rows before applying the filter.

Filter uses the current filter criteria for the DataWindow. To change the filter criteria, use the SetFilter method. The SetFilter method is equivalent to using the Filter command on the Rows menu of the DataWindow painter. If you do not call SetFilter to assign or change criteria before calling the Filter method, the DataWindow will default to use the criteria in the object definition.

When the Retrieve method retrieves data for the DataWindow, PocketBuilder applies the filter that was defined for the DataWindow object, if any. You only need to call Filter after you change the filter criteria with SetFilter or if the data has changed because of processing or user input.

Filter has no effect on the DataWindows in a composite report.

Filtering and groups
When you filter a DataWindow with groups, you might need to call GroupCalc after you call Filter.

For information on removing the filter or letting the user specify a filter expression, see SetFilter.

Examples

This statement displays rows in dw_Employee based on its current filter criteria:

```
dw_Employee.SetRedraw(false)
dw_Employee.Filter()
dw_Employee.SetRedraw(true)
```

See also

FilteredCount
RowCount
SetFilter

FilteredCount

Description

Reports the number of rows that are not displayed in the DataWindow because of the current filter criteria.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

long *dwcontrol*.**FilteredCount** ( )

Argument	Description
dwcontrol	A reference to a DataWindow control, DataStore, or child DataWindow

Return value

Returns the number of rows in *dwcontrol* that are not displayed because they do not meet the current filter criteria. Returns 0 if all rows are displayed and -1 if an error occurs. If *dwcontrol* is NULL, the method returns NULL.

Usage

A DataWindow object can have a filter as part of its definition. After the DataWindow retrieves data, the filter is applied and rows that do not meet the filter criteria are moved to the filter buffer. You can change the filter criteria by calling the SetFilter method, and you can apply the new criteria with the Filter method.

Examples

These statements retrieve data in dw_Employee, display employees with area code 617, and then test to see if any other data was retrieved. If the filter criteria specifying the area code was part of the DataWindow definition, it would be applied automatically after calling Retrieve and you would not need to call SetFilter and Filter:

```
dw_Employee.Retrieve()
dw_Employee.SetFilter("AreaCode=617")
dw_Employee.SetRedraw(false)
dw_Employee.Filter()
dw_Employee.SetRedraw(true)

// Did any rows get filtered out
IF dw_Employee.FilteredCount() > 0 THEN
       ... // Process rows not in area code 617
END IF
```

These statements retrieve data in dw_Employee and display the number of employees whose names do not begin with B:

```
dw_Employee.Retrieve()
```

```
dw_Employee.SetFilter("Left(emp_lname, 1)=~"B~"")
dw_Employee.SetRedraw(false)
dw_Employee.Filter()
dw_Employee.SetRedraw(true)

IF dw_Employee.FilteredCount() > 0 THEN
       MessageBox("Employee Count", &
           String(dw_Employee.FilteredCount()) + &
           "Employee names do not begin with B.")
END IF
```

See also

Filter
ModifiedCount
RowCount
SetFilter

Find

Description

Finds the next row in a DataWindow or DataStore in which data meets a specified condition.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

long *dwcontrol*.**Find** (string *expression*, long *start*, long *end*)

Argument	Description
dwcontrol	A reference to the DataWindow control, DataStore, or child DataWindow in which you want to search the detail band.
expression	A string whose value is a boolean expression that you want to use as the search criterion. The expression includes column names.
start	A value identifying the row location at which to begin the search. *Start* can be greater than the number of rows.
end	A value identifying the row location at which to end the search. *End* can be greater than the number of rows. To search backward, make *end* less than *start*.

Return value

Returns the number of the first row that meets the search criteria within the search range. Returns 0 if no rows are found and one of these negative numbers if an error occurs:

-1 General error
-5 Bad argument

If any argument's value is NULL, the method returns NULL.

Usage

The search is case sensitive. When you compare text to a value in a column, the case must match.

When the Find expression includes quotes If the text you want to find includes quotes, you must treat the nested quote as doubly nested, because the DataWindow parses the string twice before the Find method uses it. Therefore, you cannot simply alternate double and single quotes, as you can in most strings.

For example, to find the name O'Connor, the Find expression can be:

`"O~~~'Connor"` (3 tildes and single quote) or
`"O~~~~~"Connor"` (5 tildes and double quote)

but not:

`"O'Connor"` or `"O~"OConnor"`

When the last row satisfies the search criteria If you use Find in a loop that searches through all rows, you may end up with an endless loop if the last row satisfies the search criteria. When the *start* value becomes greater than *end*, the search reverses direction and Find would always succeed, resulting in an endless loop.

To solve this problem, you could make the *end* value 1 greater than the number of rows (see the examples). Another approach, shown below, would be to test within the loop whether the current row is greater than the row count and, if so, exit. The following code illustrates how:

```
long ll_find = 1, ll_end
ll_end = dw_main.RowCount()
ll_find = dw_main.Find(searchstr, ll_find, ll_end)
DO WHILE ll_find > 0
       ... // Collect found row
       ll_find++
       // Prevent endless loop
       IF ll_find > ll_end THEN EXIT
       ll_find = dw_main.Find(searchstr, ll_find,
ll_end)
LOOP
```

Examples

This statement searches for the first row in dw_status in which the value of the emp_salary column is greater than 100,000. The search begins in row 3 and continues until it reaches the last row in dw_status:

```
long ll_found
ll_found = dw_status.Find("emp_salary > 100000", &
       3, dw_status.RowCount())
```

To test values in more than one column, use boolean operators to join conditional expressions. The following statement searches for the employee named Smith whose salary exceeds 100,000:

```
long ll_found
ll_found = dw_status.Find( &
       "emp_lname = 'Smith' and emp_salary > 100000", &
       1, dw_status.RowCount())
```

These statements search for the first row in dw_emp that matches the value that a user entered in the SingleLineEdit called Name (note the single quotes embedded in the search expression around the name):

```
string ls_lname_emp
long ll_nbr, ll_foundrow

ll_nbr = dw_emp.RowCount()

// Remove leading and trailing blanks.
ls_lname_emp = Trim(sle_Name.Text)

ll_foundrow = dw_emp.Find( &
       "emp_lname = '" + ls_lname_emp + "'", 1, ll_nbr)
```

This script excerpt finds the first row that has a null value in emp_id. If no null is found, the script updates the DataWindow object. If a null is found, it displays a message:

```
IF dw_status.AcceptText() = 1 THEN
       IF dw_status.Find("IsNull(emp_id)", &
              1, dw_status.RowCount()) > 0 THEN
          MessageBox("Caution", "Cannot Update")
       ELSE
          dw_status.Update()
       END IF
END IF
```

The following script attached to a Find Next command button searches for the next row that meets the specified criteria and scrolls to that row. Each time the button is clicked, the number of the found row is stored in the instance variable il_found. The next time the user clicks Find Next, the search continues from the following row. When the search reaches the end, a message tells the user that no row was found. The next search begins again at the first row.

Note that although the search criteria are hard-coded here, a more realistic scenario would include a Find button that prompts the user for search criteria. You could store the criteria in an instance variable, which Find Next could use:

```
long ll_row

// Get the row num. for the beginning of the search
// from the instance variable, il_found
ll_row = il_found

// Search using predefined criteria
ll_row = dw_main.Find( &
       "item_id = 3 or item_desc = 'Nails'", &
       ll_row, dw_main.RowCount())

IF ll_row > 0 THEN
       // Row found, scroll to it and make it current
       dw_main.ScrollToRow(ll_row)
ELSE
       // No row was found
       MessageBox("Not Found", "No row found.")
END IF

// Save the number of the next row for the start
// of the next search. If no row was found,
// ll_row is 0, making il_found 1, so that
// the next search begins again at the beginning
il_found = ll_row + 1
```

This example searches all the rows in dw_main and builds a list of the names that include a lowercase a. Note that the end value of the search is one greater than the row count, avoiding an infinite loop if the name in the last row satisfies the search:

```
long ll_find, ll_end
string ll_list

// The end value is one greater than the row count
ll_end = dw_main.RowCount() + 1
ll_find = 1
```

```
ll_find = dw_main.Find("Pos(last_name,'a') > 0", &
       ll_find, ll_end)

DO WHILE ll_find > 0
       //collect names
       ll_list = ll_list + '~r' &
          + dw_main.GetItemString(ll_find,'last_name')

       // Search again
       ll_find++
       ll_find = dw_main.Find("Pos(last_name,'a') >
0",&
          ll_find, ll_end )
LOOP
```

See also

FindGroupChange
FindRequired

FindGroupChange

Description

Searches for the next break for the specified group. A group break occurs when the value in the column for the group changes. FindGroupChange reports the row that begins the next section.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

long *dwcontrol*.**FindGroupChange** (long *row*, integer *level*)

Argument	Description
dwcontrol	A reference to a DataWindow control or the DataStore.
row	A value identifying the row at which you want to begin searching for the group break.
level	The number of the group for which you are searching. Groups are numbered in the order in which you defined them.

Return value

Returns the number of the row whose group column has a new value, meaning that it begins a new group. Returns 0 if the value in the group column did not change and a negative number if an error occurs. If any argument's value is NULL, the method returns NULL.

The return value observes these rules based on the value of *row*. If the starting row is:

- The first row in a group, then FindGroupChange returns the starting row number
- A row within a group, other than the last group, then FindGroupChange returns the row number of the first row of the next group
- A row in the last group, other than the first row of the last group, then FindGroupChange returns 0

Usage

If the starting row begins a new section at the specified level, then that row is the one returned. To continue searching for subsequent breaks, increment the starting row so that the search resumes with the second row in the group.

Examples

This statement searches for the first break in group 2 in dw_regions. The search begins in row 5:

```
dw_regions.FindGroupChange(5, 2)
```

This code finds the number of the row at which a break occurs in group 1. It then checks whether the department number is 121. The search begins at row 0:

```
boolean lb_found
long ll_breakrow

lb_found = FALSE
ll_breakrow = 0

DO WHILE NOT (lb_found)
    ll_breakrow = dw_1.FindGroupChange(ll_breakrow, 1)

    // If no breaks are found, exit.
    IF ll_breakrow <= 0 THEN EXIT

    // Have we found the section for Dept 121?
    IF dw_1.GetItemNumber(ll_breakrow, &
       "dept_id") = 121 THEN
          lb_found = TRUE
    END IF

    // Increment starting row to find next break
    ll_breakrow = ll_breakrow + 1
LOOP

IF lb_found = FALSE THEN
    MessageBox( &
       "Not Found", &
```

```
        "The Department was not found.")
ELSE
        ... // Processing for Dept 121
END IF
```

See also

Find
FindRequired

FindNext

Description

Finds the next occurrence of text in a RichTextEdit DataWindow control and highlights it, using criteria set up in a previous call of the Find method.

PocketBuilder	✗
PowerBuilder	✓

Syntax

PowerBuilder

integer *dwcontrol*.**FindNext** ( )

Return value

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.

FindRequired

Description

Reports the next row and column that is required and contains a NULL value. The method arguments that specify where to start searching also store the results of the search. You can speed up the search by specifying that FindRequired check only inserted and modified rows.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

integer *dwcontrol*.**FindRequired** (DWBuffer *dwbuffer*, long *row*, integer *colnbr*, string *colname*, boolean *updateonly*)

Argument	Description
dwcontrol	A reference to the DataWindow control or DataStore in which you want to find required columns that have NULL values.

Argument	Description
dwbuffer	A value indicating the DataWindow buffer you want to search for required columns. Valid buffers are: • Primary! • Filter!
row	A value identifying the first row to be searched. Row also stores the number of the found row. FindRequired increments the row number automatically after it validates each row's columns. When it finds a row with a required column that contains a NULL value, the row number is stored in *row*. After FindRequired validates the last column in the last row, it sets *row* to 0. The *row* argument must be a variable so it can return a value for the found row.
colnbr	A value identifying the first column to be searched. *Colnbr* also stores the number of the found column. After validating the last column, FindRequired sets *colnbr* to 1 and increments *row*. When it finds a required column that contains a NULL value, the column number is stored in *colnbr*. The *colnbr* argument must be a variable so it can return a value for the found column.
colname	A string in which you want to store the name of the required column that contains a NULL value (the name of *colnbr*). The *colname* argument must be a variable so it can hold a value for the name of the found column.
updateonly	A value indicating whether you want to validate all rows and columns or only rows that have been inserted or modified: • TRUE — Validate only those rows that have changed. Setting *updateonly* to TRUE enhances performance in large DataWindows. • FALSE — Validate all rows and columns.

Return value

Returns 1 if FindRequired successfully checked the rows and -1 if an error occurs.

If any argument's value is NULL, the method returns NULL.

Usage

For FindRequired to report an empty required column, the column's value must actually be NULL, not an empty string.

To make a column required, set the Required property to TRUE in a script or check the Required check box for the column in the DataWindow painter.

New rows have NULL values in their columns, unless the columns have default values. If *updateonly* is FALSE, FindRequired reports empty required columns in new rows. If *updateonly* is TRUE, FindRequired does not check new rows because new, empty rows are not updated in the database.

When the user modifies a row and leaves a column empty, the new value is an empty string, unless the column's edit style has the Empty String Is NULL check box checked. FindRequired does not report empty required columns in modified rows unless this property is set.

Examples

The following code makes a list of all the row numbers and column names in dw_1 in which required columns are missing values. The list is displayed in the MultiLineEdit mle_required:

```
long ll_row = 1
integer colnbr = 0
string colname

mle_required.Text = ""
DO WHILE ll_row <> 0
       colnbr++ // Continue searching at next column
       // If there's an error, exit
       IF dw_1.FindRequired(Primary!, &
          ll_row, colnbr, &
          colname, FALSE) < 0 THEN EXIT

       // If a row was found, save the row and column
       IF ll_row <> 0 THEN
          mle_required.Text = mle_required.Text &
              + String(ll_row) + "~t" &
              + colname + "~r~n"
       END IF

       // When FindRequired returns 0 (meaning
       // no more rows found), drop out of loop
LOOP
```

This example is a function that ensures that no required column in a DataWindow control is empty (contains NULL). It takes one argument—the DataWindow control, which is declared in the function declaration like this:

```
DataWindow adw_control
```

The function returns -2 if the user's last entry cannot be accepted or if FindRequired returns an error. It returns -1 if an empty required column is found. It returns 1 if all required columns have data:

```
integer li_colnbr = 1
```

```
long ll_row = 1
string ls_colname, ls_textname

// Make sure the last entry is accepted
IF adw_control.AcceptText() = -1 THEN
       adw_control.SetFocus()
       RETURN -2
END IF

// Find the first empty row and column, if any
IF adw_control.FindRequired(Primary!, ll_row, &
          li_colnbr, ls_colname, true) < 1 THEN
       //If search fails due to error, then return
       RETURN -2
END IF

// Was any row found?
IF ll_row <> 0 THEN
       // Get the text of that column's label.
       ls_textname = ls_colname + "_t.Text"
       ls_colname = adw_control.Describe(ls_textname)

       // Tell the user which column to fill in
       MessageBox("Required Value Missing", &
          "Please enter a value for '" &
          + ls_colname + "', row " &
          + String(ll_row) + ".", &
          StopSign! )

       // Make the problem column current.
       adw_control.SetColumn(li_colnbr)
       adw_control.ScrollToRow(ll_row)
       adw_control.SetFocus()
       RETURN -1
END IF

// Return success code if all required
// rows and columns have data
RETURN 1
```

See also

Find
FindGroupChange
FindRequiredColumn
FindRequiredColumnName
FindRequiredRow
ScrollToRow

SetColumn
SetTransObject

FindRequiredColumn

Description

Returns the column number that the FindRequired method found. The column is being reported because it is a required column but contains a NULL value. You must call FindRequired first to search for the required but missing information.

PocketBuilder	✗
PowerBuilder	✓

Syntax

Web ActiveX

number *dwcontrol*.**FindRequiredColumn** ( )

Argument	Description
dwcontrol	A reference to the DataWindow control for which you just called FindRequired

Return value

Returns the number of a column in the DataWindow.

FindRequiredColumnName

Description

Returns the column name that the FindRequired method found. The column is being reported because it is a required column but contains a NULL value. You must call FindRequired first to search for the required but missing information.

PocketBuilder	✗
PowerBuilder	✓

Syntax

Web ActiveX

string *dwcontrol*.**FindRequiredColumnName** ( )

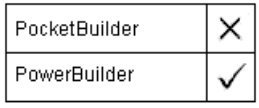
Return value

Returns the name of a column in the DataWindow.

FindRequiredRow

Description

Returns the row number that the FindRequired method found. The row is being reported because it contains a required column that has a NULL value. You must call FindRequired first to search for the required but missing information.



Syntax

Web ActiveX

number *dwcontrol*.**FindRequiredRow** ( )

Return value

Returns the number of a row in the DataWindow.

Generate

Description

Creates HTML syntax for the Web DataWindow.

PocketBuilder	✗
PowerBuilder	✓

Syntax

Web DataWindow server component

string *dwcontrol*.**Generate** ( )

Return value

Returns an HTML rendering of the current page of the DataWindow if the method succeeds and an empty string if an error occurs.

GenerateHTMLForm

Description

Creates an HTML Form element containing columns for one or more rows in a DataWindow control or DataStore. This method also returns an HTML Style element containing style sheet information.

PocketBuilder	✗
PowerBuilder	✓

Obsolete method

GenerateHTMLForm is obsolete and should not be used. The Web DataWindow generator component generates HTML and JavaScript to provide data entry, validation, and other DataWindow features. For more information, see the *DataWindow Programmer's Guide* and *Working with Web and JSP Targets* in the PowerBuillder documentation set.

Syntax

PowerBuilder

integer *dwcontrol*.**GenerateHTMLForm** (string *syntax*, string *style*, string *action* { , long *startrow*, long *endrow*, integer *startcolumn*, integer *endcolumn* {, DWBuffer *buffer* } })

Return value

Returns 1 if the method succeeds and -1 if an error occurs. If any argument is NULL, the method returns NULL.

GenerateResultSet

Generates a result set that can be used by non-DataWindow controls for displaying data. A result set is usually generated by a component on a transaction server and returned to a client application.

To generate a result set	Use
That can be an EAServer result set or an ADO Recordset	Syntax 1
Using an EAServer Method As Stored Procedure (MASP)	Syntax 2

Syntax 1 For generating an EAServer result set or an ADO Recordset

Description

Generates a result set from data in a DataStore or DataWindow control.

PocketBuilder	✗
PowerBuilder	✓

Syntax

PowerBuilder DataStore object

integer *dsobject*.**GenerateResultSet** (REF ResultSet *rsdest* { ,dwBuffer *dwbuffer* })

Return value

Returns 1 if it succeeds and -1 if it fails. If any argument is NULL, it returns NULL.

Syntax 2 For generating a result set using an EAServer Method As Stored Procedure

Description

Generates an EAServer result set that can be returned from a PowerBuilder user object running as a component on EAServer. The result set is retrieved using a DataWindow control or DataStore object whose data source is an EAServer component method.

PocketBuilder	✗
PowerBuilder	✓

Syntax

PowerBuilder DataWindow control or DataStore object

dwcontrol.**GenerateResultSet** ({ *dwbuffer* })

Return value

Returns 1 if it succeeds or a negative value if an error occurs.

GetBandAtPointer

Description

Reports the band in which the pointer is currently located, as well as the row number associated with the band. The bands are the headers, trailers, and detail areas of the DataWindow and correspond to the horizontal areas of the DataWindow painter.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

string *dwcontrol*.**GetBandAtPointer** ( )

Argument	Description
dwcontrol	A reference to a DataWindow control.

Return value

Returns a string that names the band in which the pointer is located, followed by a tab character and the number of the row associated with the band (see the table in Usage). Returns the empty string ("") if an error occurs. If *dwcontrol* is NULL, the method returns NULL.

Usage

The following table lists the band names, where the pointer is when a given band is reported, and the row that is associated with the band.

Band	Location of pointer	Associated row
detail	In the body of the DataWindow object	The row at the pointer. If rows do not fill the body of the DataWindow object because of a group with a page break, then the first row of the next group. If the body is not filled because there are no more rows, then the last row.
header	In the header of the DataWindow object	The first row visible in the DataWindow body.
header.n	In the header of group level n	The first row of the group.
trailer.n	In the trailer of group level n	The last row of the group.
footer	In the footer of the DataWindow object	The last row visible in the DataWindow body.
summary	In the summary of the DataWindow object	The last row before the summary.

You can parse the return value by searching for the tab character (ASCII 09). In PocketBuilder, search for ~t. For an example that parses a string that includes a tab, see GetValue on page 516.

Examples

These statements set the string named band to the location of the pointer in DataWindow dw_rpt:

```
String band
band = dw_rpt.GetBandAtPointer()
```

Some possible return values are:

Table 9-3: Example return values for the GetBandAtPointer method

Return value	Meaning
detail[tab]8	In row 8 of the detail band of dw_rpt
header[tab]10	In the header of dw_rpt; row 10 is the first visible row
header.2[tab]1	In the header of group level 2 for row 1
trailer.1[tab]5	In the trailer of group level 1 for row 5
footer[tab]111	In the footer of dw_rpt; the last visible row is 111
summary[tab]23	In the summary of dw_rpt; the last row is 23

See also

GetObjectAtPointer

GetBorderStyle

Description

Reports the border style of a column in a DataWindow control or DataStore object.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

border *dwcontrol*.**GetBorderStyle** (integer *column*)

border *dwcontrol*.**GetBorderStyle** (string *column*)

Argument	Description
dwcontrol	A reference to a DataWindow control, DataStore, or child DataWindow.
column	The column for which you want to obtain the border style. *Column* can be a column number or a column name.

Return value

Returns the border style of *column* in *dwcontrol* as a value of the Border enumerated datatype. For a list of possible values, see Border on page 369. Returns NULL if it fails. If any argument is NULL, the method returns NULL.

Examples

This code gets the border style for the current column:

```
border B2
B2 = dw_emp.GetBorderStyle(dw_emp.GetColumn())
```

This code tests the border of column 2 in dw_emp and, if there is no border, displays a shadow box border:

```
border B2
B2 = dw_emp.GetBorderStyle(2)
IF B2 = NoBorder! THEN
       dw_emp.SetBorderStyle(2, ShadowBox!)
END IF
```

See also

SetBorderStyle

GetChanges

Description

Retrieves changes made to a DataWindow or DataStore as a blob. This method is used primarily in distributed applications.

PocketBuilder	✗
PowerBuilder	✓

Syntax

PowerBuilder DataWindow control or DataStore object

long *dwcontrol*.**GetChanges** (REF blob *changeblob* {, blob *cookie* })

Return value

Returns the number of rows in the DataWindow change blob if it succeeds or a negative value if it fails.

GetChangesBlob

Description

Returns changes made to a DataWindow or DataStore. You must call GetChanges first to set up the change information. This method is used primarily in distributed applications.

PocketBuilder	✗
PowerBuilder	✓

Syntax

Web ActiveX

string *dwcontrol*.**GetChangesBlob** ( )

Return value

Returns a string whose value is the DataWindow change blob set up by GetChanges. If *dwcontrol* is NULL, the method returns NULL.

GetChild

Description

Provides a reference to a child DataWindow or to a report in a composite DataWindow that you can use in DataWindow functions to manipulate that DataWindow or report.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

integer *dwcontrol*.**GetChild** (string *name*, REF DataWindowChild *dwchildvariable*)

Argument	Description
dwcontrol	A reference to the DataWindow control or DataStore that contains the child DataWindow or report.
name	A string that names the column containing the child DataWindow or that names the report in the composite DataWindow.
dwchildvariable	A variable in which you want to store the reference to the child DataWindow or report. (For the PowerBuilder Web ActiveX, the separate function GetChildObject must be called to get the reference variable to the child object.)

Return value

Returns 1 if it succeeds and -1 if an error occurs—for example, if the child object does not exist. If any argument is NULL, the method returns NULL.

Usage

A child DataWindow is a DropDownDataWindow in a DataWindow object.

A report is a DataWindow that is part of a composite DataWindow. A report is read-only. When you define the composite DataWindow in the DataWindow painter, each report is given a name. You can see the name in the Name option of the Properties view. You must use the report name (not the name of the DataWindow object in which the report has been placed) when calling GetChild.

Use GetChild when you need to explicitly retrieve data for a child DataWindow or report. Although PocketBuilder automatically retrieves data for the child or report when the main DataWindow is displayed, you need to explicitly retrieve data when there are retrieval arguments or when conditions change and you want to retrieve new rows.

When you insert a row or retrieve data in the main DataWindow, PocketBuilder automatically retrieves data for the child DataWindow. If the child DataWindow has retrieval arguments, PocketBuilder displays a dialog box asking the user for values for those arguments. To suppress the dialog box, you can explicitly retrieve data for the child before changing the main DataWindow (see the example).

Changing property values with the Modify method can cause the reference returned by GetChild to become invalid. After setting such a property, call GetChild again. If a property causes this behavior, this is noted in its description in Chapter 3, “DataWindow Object Properties.”

Examples

This example retrieves data for the child DataWindow associated with the column emp_state before retrieving data in the main DataWindow. The child DataWindow expects a region value as a retrieval argument. Because you populate the child DataWindow first, specifying a value for its retrieval argument, there is no need for PocketBuilder to display the retrieval argument dialog box:

```
DataWindowChild state_child
integer rtncode

rtncode = dw_1.GetChild('emp_state', state_child)
IF rtncode = -1 THEN MessageBox( &
       "Error", "Not a DataWindowChild")

// Establish the connection
CONNECT USING SQLCA;

// Set the transaction object for the child
state_child.SetTransObject(SQLCA)
```

```
// Populate with values for eastern states
state_child.Retrieve("East")

// Set transaction object for main DW and retrieve
dw_1.SetTransObject(SQLCA)
dw_1.Retrieve()
```

In a composite DataWindow there are two reports: orders and current inventory. The orders report has a retrieval argument for selecting the order status. This report displays open orders. The composite DataWindow is displayed in a DataWindow control called dw_news and the reports are named open_orders and current_inv. The following code in the Open event of the window that contains dw_news provides a retrieval argument for open_orders:

```
DataWindowChild dwc_orders
dw_news.GetChild("open_orders", dwc_orders)
dwc_orders.SetTransObject(SQLCA)
dwc_orders.Retrieve("open")
```

See also

GetChildObject
SetTransObject

GetChildObject

Description

Gets the reference to a child object for a Web ActiveX DataWindow.

Syntax

Web ActiveX

OleObject *dwcontrol*.**GetChildObject** ( )

Return value

Returns an object that is the DataWindowChild or report. If no object is found, a null object reference is returned.

GetClickedColumn

Description

Obtains the number of the column the user clicked or double-clicked in a DataWindow control or DataStore object.

PocketBuilder	✗
PowerBuilder	✓

Syntax

PowerBuilder DataWindow control, DataStore object, or child DataWindow

integer *dwcontrol*.**GetClickedColumn** ( )

Return value

Returns the number of the column that the user clicked or double-clicked in *dwcontrol*. Returns 0 if the user did not click or double-click a column (for example, the user double-clicked outside the data area, in text or spaces between columns, or in the header, summary, or footer area). If *dwcontrol* is NULL, the method returns NULL.

GetClickedRow

Description

Obtains the number of the row the user clicked or double-clicked in a DataWindow control or DataStore object.

PocketBuilder	✗
PowerBuilder	✓

Syntax

PowerBuilder DataWindow control or DataStore object

long *dwcontrol*.**GetClickedRow** ( )

Return value

Returns the number of the row that the user clicked or double-clicked in *dwcontrol*. Returns 0 if the user did not click or double-click a row (for example, the user double-clicked outside the data area, in text or spaces between rows, or in the header, summary, or footer area). If *dwcontrol* is NULL, the method returns NULL.

GetColumn

Description

Obtains the number of the current column. The current column is the column that has focus.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

integer *dwcontrol*.**GetColumn** ( )

Argument	Description
dwcontrol	A reference to a DataWindow control DataStore, or child DataWindow

Return value

Returns the number of the current column in *dwcontrol*. Returns 0 if no column is current (because all the columns have a tab value of 0, making all of them uneditable), and -1 if an error occurs. If *dwcontrol* is NULL, the method returns NULL.

Usage

GetColumn and GetClickedColumn, when called in the Clicked or DoubleClicked event, can return different values. The column the user clicked doesn't become current until after the event.

Use GetColumnName (instead of GetColumn) when you need the column's name. Use SetColumn to change the current column.

Using with other controls

For use with ListView controls, see GetColumn in the *PowerScript Reference*.

The current column

A column becomes the current column after the user tabs to it or clicks it or if a script calls the SetColumn method. A column cannot be current if it cannot be edited (if it has a tab value of 0).

A DataWindow always has a current column, even when the control is not active, as long as there is at least one editable column.

Examples

These statements return the number of the current column in dw_Employee:

```
integer li_ColNum
li_ColNum = dw_employee.GetColumn()
```

See also

GetClickedColumn
GetColumnName
GetRow
SetColumn
SetRow

GetColumnName

Description

Obtains the name of the current column. The current column is the column that has the focus.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

string *dwcontrol*.**GetColumnName** ( )

Argument	Description
dwcontrol	A reference to a DataWindow control DataStore, or child DataWindow

Return value

Returns the name of the current column in *dwcontrol*. Returns the empty string ("") if no column is current or if an error occurs. If *dwcontrol* is NULL, the method returns NULL.

Usage

For information on the current column, see GetColumn on page 480.

Examples

These statements return the name of the current column in dw_Employee:

```
string ls_ColName
ls_ColName = dw_employee.GetColumnName()
```

See also

GetColumn
GetRow
SetColumn
SetRow

GetContextService

Description

Creates a reference to a context-specific instance of the specified service.

PocketBuilder	✗
PowerBuilder	✓

Syntax

PowerBuilder DataWindow control, DataStore object, or child DataWindow

integer *objectname*.**GetContextService** (string *servicename*, PowerObject *servicereference*)

Return value

Returns 1 if the method succeeds and -1 if an error occurs.

GetFormat

Description

Obtains the display format assigned to a column in a DataWindow control or DataStore object.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

string *dwcontrol*.**GetFormat** (string *column*)

string *dwcontrol*.**GetFormat** (integer *column*)

Argument	Description
dwcontrol	A reference to a DataWindow control, DataStore, or child DataWindow.
column	The column for which you want the display format. *Column* can be a column number (integer) or a column name (string).

Return value

Returns the display format specification for *column* in *dwcontrol*. If an error occurs, GetFormat returns the empty string (""). If any argument value is NULL, the method returns NULL.

Usage

If you want to change the display format of a column temporarily, you can use GetFormat to save the current format.

Examples

These statements save the format of column salary of dw_employee before changing it to a new format:

```
string OldFormat, NewFormat = "$##,###.00"
OldFormat = dw_employee.GetFormat("salary")
dw_employee.SetFormat("salary", NewFormat)
```

See also

SetFormat

GetFullContext

Description

This method returns a string representing the context of the client-side control to be passed on a form submit.

PocketBuilder	✗
PowerBuilder	✓

Syntax

Web DataWindow client control

string *dwcontrol*.GetFullContext ( )

Return value

String

GetFullState

Description

Retrieves the complete state of a DataWindow or DataStore as a blob.

This method is used primarily in distributed applications.

PocketBuilder	✗
PowerBuilder	✓

Syntax

PowerBuilder DataWindow control or DataStore object

long *dwcontrol*.**GetFullState** (blob *dwasblob*)

Return value

Returns the number of rows in the DataWindow blob if it succeeds and -1 if an error occurs. GetFullState will return -1 if the DataWindow control or DataStore does not have a DataWindow object associated with it. If any argument value is NULL, the method returns NULL.

GetFullStateBlob

Description

Returns the state of a DataWindow or DataStore. You must call GetFullState first to set up the state information. This method is used primarily in distributed applications.

PocketBuilder	✗
PowerBuilder	✓

Syntax

Web ActiveX

string *dwcontrol*.**GetFullStateBlob** ( )

Return value

Returns a string whose value is the DataWindow state blob set up by GetFullState. If *dwcontrol* is NULL, the method returns NULL.

GetItem

Description

Gets the value of an item for the specified row and column. GetItem returns the value available in the data available to the client. This is equivalent to the primary buffer in other environments.

PocketBuilder	✗
PowerBuilder	✓

Syntax

Web DataWindow client control

returnvalue *dwcontrol*.**GetItem** (number *row*, number *column*)

returnvalue *dwcontrol*.**GetItem** (number *row*, string *column*)

Return value

Returns the value in the specified row and column. The datatype of the returned data corresponds to the datatype of the column. Returns NULL if the column value is NULL. Returns the empty string ("") if an error occurs. If any argument value is NULL, the method returns NULL.

GetItemDate

Description

Gets data whose type is Date from the specified buffer of a DataWindow control or DataStore object. You can obtain the data that was originally retrieved and stored in the database from the original buffer, as well as the current value in the primary, delete, or filter buffers.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

date *dwcontrol*.**GetItemDate** (long *row*, string *column* {, DWBuffer *dwbuffer* , boolean *originalvalue* })

date *dwcontrol*.**GetItemDate** (long *row*, integer *column* {, DWBuffer *dwbuffer*, boolean *originalvalue* })

Argument	Description
dwcontrol	A reference to a DataWindow control, DataStore, or child DataWindow.
row	A value identifying the row location of the data.
column	The column location of the data. The datatype of the column must be date. *Column* can be a column number or a column name. The column number is the number of the column as it is listed in the Column Specification view of the DataWindow painter—not necessarily the number of the column in the Design view. To get the contents of a computed field, specify the name of the computed field for *column*. Computed fields do not have numbers.
dwbuffer (optional)	A value identifying the DataWindow buffer from which you want to get the data. For a list of valid values, see DWBuffer on page 372.
originalvalue (optional)	A boolean indicating whether you want the original or current values for *row* and *column*: • True — Returns the original values (the values initially retrieved from the database). • False — (Default) Returns the current values. If you specify *dwbuffer*, you must also specify *originalvalue*.

Return value

Returns the date value in the specified row and column. Returns NULL if the column value is NULL or if there is no DataWindow object assigned to the DataWindow control or DataStore. Returns 1900-01-01 if any other error occurs. If any argument value is NULL, the method returns NULL.

Usage

Use GetItemDate when you want to get information from the DataWindow's buffers. To find out what the user entered in the current column before that data is accepted, use GetText. In the ItemChanged or ItemError events, use the data argument.

To access a row in the original buffer, specify the buffer that the row currently occupies (primary, delete, or filter) and the number of the row in that buffer. When you specify TRUE for *originalvalue*, the method gets the original data for that row from the original buffer.

An execution error occurs when the datatype of the DataWindow column does not match the datatype of the method; in this case, date.

Datatypes of columns and computed fields
There is a difference in datatypes between columns and computed columns retrieved from the database and computed fields defined in the DataWindow painter. Computed columns from the database can have a datatype of date, but a date computed field always has a datatype of DateTime, not date. In PocketBuilder, use the GetItemDateTime method instead.

Using GetItemDate in a String function
When you call GetItemDate 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 "2/26/96 00:00:00":

```
String(dw_1.GetItemDate(1, "start_date"))
```

To get a simple date string, you can specify a display format:

```
String(dw_1.GetItemDate(1,"start_date"), "m/d/yy")
```

or you can assign the date to a date variable before calling the String function:

```
date ld_date
string ls_date
ld_date = dw_1.GetItemDate(1, "start_date")
ls_date = String(ld_date)
```

Examples

These statements set hiredate to the current Date data in the third row of the primary buffer in the column named first_day of dw_employee:

```
Date hiredate
hiredate = dw_employee.GetItemDate(3, "first_day")
```

These statements set hiredate to the current Date data in the third row of the filter buffer in the column named first_day of dw_employee:

```
Date hiredate
hiredate = dw_employee.GetItemDate(3, &
       "first_day", Filter!, FALSE)
```

These statements set hiredate to original Date data in the third row of the primary buffer in the column named hdate of dw_employee:

```
Date hiredate
hiredate = dw_employee.GetItemDate(3, &
       "hdate", Primary!, TRUE)
```

See also

GetItemDateTime
GetItemDecimal

GetItemNumber
GetItemString
GetItemTime
GetText
SetItem
SetText

GetItemDateTime

Description

Gets data whose type is DateTime from the specified buffer of a DataWindow control or DataStore object. You can obtain the data that was originally retrieved and stored in the database from the original buffer, as well as the current value in the primary, delete, or filter buffers.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

DateTime *dwcontrol*.**GetItemDateTime** (long *row*, string *column* {, DWBuffer *dwbuffer*, boolean *originalvalue* })

DateTime *dwcontrol*.**GetItemDateTime** (long *row*, integer *column* {, DWBuffer *dwbuffer*, boolean *originalvalue* })

Argument	Description
dwcontrol	A reference to the DataWindow control, DataStore, or child DataWindow in which you want to obtain the DateTime data contained in a specific row and column.
row	A value identifying the row location of the data.
column	The column location of the data. The datatype of the column must be DateTime. *Column* can be a column number or a column name. The column number is the number of the column as it is listed in the Column Specification view of the DataWindow painter—not necessarily the number of the column in the Design view. To get the contents of a computed field, specify the name of the computed field for *column*. Computed fields do not have numbers.
dwbuffer (optional)	A value identifying the DataWindow buffer from which you want to get the data. For a list of valid values, see DWBuffer on page 372.

Argument	Description
originalvalue (optional)	A boolean indicating whether you want the original or current values for *row* and *column*: • True — Returns the original values, that is, the values initially retrieved from the database. • False — (Default) Returns the current values. If you specific *dwbuffer*, you must also specify *originalvalue*.

Return value

Returns the DateTime or Timestamp value in the specified row and column. Returns NULL if the column value is NULL or if there is no DataWindow object assigned to the DataWindow control or DataStore. Returns 1900-01-01 00:00:00.000000 if any other error occurs. If any argument value is NULL, the method returns NULL.

Usage

Use GetItemDateTime when you want to get information from the DataWindow's buffers. To find out what the user entered in the current column before that data is accepted, use GetText. In the ItemChanged or ItemError events, use the data argument.

To access a row in the original buffer, specify the buffer that the row currently occupies (primary, delete, or filter) and the number of the row in that buffer. When you specify TRUE for *originalvalue*, the method gets the original data for that row from the original buffer.

Datatype mismatch
An execution error occurs when the datatype of the DataWindow column does not match the datatype of the method—in this case, DateTime.

Computed fields displaying date or time values have a datatype of DateTime, not date or time. Always use GetItemDateTime to get their value, not GetItemDate or GetItemTime.

Examples

These statements set as_of to the current DateTime data in the primary buffer for row 3 of the column named start_dt in the DataWindow dw_emp:

```
DateTime as_of
as_of = dw_emp.GetItemDateTime(3, "start_dt")
```

These statements set as_of to the current DateTime data in the delete buffer for row 3 of the end_dt column of dw_emp:

```
DateTime as_of
as_of = dw_emp.GetItemDateTime(3, "end_dt", &
       Delete!, false)
```

These statements set AsOf to the original DateTime data in the primary buffer for row 3 of the end_dt column of dw_emp:

```
DateTime as_of
as_of = dw_emp.GetItemDateTime(3, "end_dt", &
       Primary!, TRUE)
```

See also

GetItemDate
GetItemDecimal
GetItemNumber
GetItemString
GetItemTime
SetItem

GetItemDecimal

Description

Gets data whose type is decimal from the specified buffer of a DataWindow control or DataStore object. You can obtain the data that was originally retrieved and stored in the database from the original buffer, as well as the current value in the primary, delete, or filter buffers.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

decimal *dwcontrol*.**GetItemDecimal** (long *row*, integer *column* {, DWBuffer *dwbuffer*, boolean *originalvalue* })

decimal *dwcontrol*.**GetItemDecimal** (long *row*, string *column* {, DWBuffer *dwbuffer*, boolean *originalvalue* })

Argument	Description
dwcontrol	A reference to a DataWindow control or DataStore.
row	A value identifying the row location of the decimal data.
column	The column location of the data. The datatype of the column must be one of type decimal. *Column* can be a column number or a column name. The column number is the number of the column as it is listed in the Column Specification view of the DataWindow painter—not necessarily the number of the column in the Design view. To get the contents of a computed field, specify the name of the computed field for *column*. Computed fields do not have numbers.

Argument	Description
dwbuffer (optional)	A value of the dwBuffer enumerated datatype identifying the DataWindow buffer from which you want to get the data. For a list of valid values, see DWBuffer on page 372.
originalvalue (optional)	A boolean indicating whether you want the original or current values for *row* and *column*: • True — Returns the original values, that is, the values initially retrieved from the database. • False — (Default) Returns the current values. If you specify *dwbuffer*, you must also specify *originalvalue*.

Return value

Returns the decimal value in the specified row and column. Returns NULL if the column value is NULL or if there is no DataWindow object assigned to the DataWindow control or DataStore. Triggers the SystemError event and returns -1 if any other error occurs (see "Handling errors" below). If any argument value is NULL, the method returns NULL.

Usage

Use GetItemDecimal when you want to get information from the DataWindow's buffers. To find out what the user entered in the current column before that data is accepted, use GetText. In the ItemChanged or ItemError events, use the data argument.

To access a row in the original buffer, specify the buffer that the row currently occupies (primary, delete, or filter) and the number of the row in that buffer. When you specify TRUE for *originalvalue*, the method gets the original data for that row from the original buffer.

Handling errors

The return value is a valid value from the database unless the SystemError event is triggered. When the value cannot be converted because the column's datatype does not match the method's datatype, an execution error occurs, which triggers the SystemError event. The default error processing halts the application.

If you write a script for the SystemError event, it should also halt the application. Therefore, the error return value is seldom used.

Examples

These statements set salary_amt to the current decimal data in the primary buffer for row 4 of the column named emp_salary of dw_employee:

```
decimal salary_amt
salary_amt = &
       dw_employee.GetItemDecimal(4, "emp_salary")
```

These statements set salary_amt to the current decimal data in the filter buffer for row 4 of the column named emp_salary of dw_employee:

```
decimal salary_amt
salary_amt = dw_employee.GetItemDecimal(4, &
       "emp_salary", Filter!, FALSE)
```

These statements set salary_amt to the original decimal data in the primary buffer for row 4 of the column named emp_salary of dw_employee:

```
decimal salary_amt
salary_amt = dw_employee.GetItemDecimal(4, &
       "emp_salary", Primary!, TRUE)
```

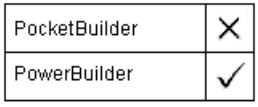
See also

GetItemDate
GetItemDateTime
GetItemNumber
GetItemString
GetItemTime
SetItem

GetItemFormattedString

Description

Gets and formats data whose type is String from the specified buffer of a DataWindow control or DataStore object.



Syntax

PowerBuilder DataWindow control, DataStore object, or child DataWindow

string *dwcontrol*.**GetItemFormattedString** (long *row*, integer *column* {, DWBuffer *dwbuffer*, boolean *originalvalue* })

string *dwcontrol*.**GetItemFormattedString** (long *row*, string *column* {, DWBuffer *dwbuffer*, boolean *originalvalue* })

Return value

String. Returns the value of the data in its current display format.

GetItemNumber

Description

Gets numeric data from the specified buffer of a DataWindow control or DataStore object. You can obtain the data that was originally retrieved and stored in the database from the original buffer, as well as the current value in the primary, delete, or filter buffers.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

numeric *dwcontrol*.**GetItemNumber** (long *row*, string *column* {, DWBuffer *dwbuffer*, boolean *originalvalue* })

numeric *dwcontrol*.**GetItemNumber** (long *row*, integer *column* {, DWBuffer *dwbuffer*, boolean *originalvalue* })

Argument	Description
dwcontrol	A reference to a DataWindow control, DataStore, or child DataWindow.
row	A value identifying the row location of the numeric data.
column	The column location of the numeric data. The datatype of the column must be one of a numeric datatype. *Column* can be a column number or a column name. The column number is the number of the column as it is listed in the Column Specification view of the DataWindow painter—not necessarily the number of the column in the Design view. To get the contents of a computed field, specify the name of the computed field for *column*. Computed fields do not have numbers.
dwbuffer (optional)	A value identifying the DataWindow buffer from which you want to get the data. For a list of valid values, see DWBuffer on page 372.
originalvalue (optional)	A boolean indicating whether you want the original or current values for *row* and *column*: • True — Return the original values (the values initially retrieved from the database). • False — (Default) Return the current values. If you specify *dwbuffer*, you must also specify *originalvalue*.

Return value

Returns the numeric value in the specified row and column (decimal, double, integer, long, or real). Returns NULL if the column value is NULL or if there is no DataWindow object assigned to the DataWindow control or DataStore. Triggers the SystemError event and returns -1 if any other error occurs (see "Handling errors" below). If any argument value is NULL, the method returns NULL.

Usage

Use GetItemNumber to get information from the DataWindow's buffers. To find out what the user entered in the current column before that data is accepted, use GetText. In the ItemChanged or ItemError events, use the data argument.

To access a row in the original buffer, specify the buffer that the row currently occupies (primary, delete, or filter) and the number of the row in that buffer. When you specify TRUE for *originalvalue*, the method gets the original data for that row from the original buffer.

Handling errors

The return value is a valid value from the database unless the SystemError event is triggered. When the value cannot be converted because the column's datatype does not match the method's datatype, an execution error occurs, which triggers the SystemError event. The default error processing halts the application. If you write a script for the SystemError event, it should also halt the application. Therefore, the error return value is seldom used.

Examples

These statements set EmpNbr to the current numeric data in the primary buffer for row 4 of the column named emp_nbr in dw_employee:

```
integer EmpNbr
EmpNbr = dw_employee.GetItemNumber(4, "emp_nbr")
```

These statements set EmpNbr to the current numeric data in the filter buffer for row 4 of the column named salary of dw_employee:

```
integer EmpNbr
EmpNbr = dw_employee.GetItemNumber(4, &
       "salary", Filter!, FALSE)
```

These statements set EmpNbr to the original numeric data in the primary buffer for row 4 of the column named salary of dw_Employee:

```
integer EmpNbr
EmpNbr = dw_Employee.GetItemNumber(4, &
       "salary", Primary!, TRUE)
```

See also

GetItemDate
GetItemDateTime

GetItemDecimal
GetItemString
GetItemTime
SetItem

GetItemStatus

Description

Reports the modification status of a row or a column within a row. The modification status determines the type of SQL statement the Update method will generate for the row or column.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

DWItemStatus *dwcontrol*.**GetItemStatus** (long *row*, integer *column*, DWBuffer *dwbuffer*)

DWItemStatus *dwcontrol*.**GetItemStatus** (long *row*, string *column*, DWBuffer *dwbuffer*)

Argument	Description
dwcontrol	A reference to a DataWindow control, DataStore, or child DataWindow.
row	A value identifying the row for which you want the status.
column	The column for which you want the status. *Column* can be a column number or a column name. The column number is the number of the column as it is listed in the Column Specification view of the DataWindow painter—not necessarily the number of the column in the Design view. Specify 0 to get the status of the whole row.
dwbuffer	A value identifying the DataWindow buffer containing the row for which you want status. For a list of valid values, see DWBuffer on page 372.

Return value

A value of the dwItemStatus enumerated datatype. The return value identifies the status of the item at *row*, *column* of *dwcontrol* in *dwbuffer*. For a list of status values, see DWItemStatus on page 373.

If column is 0, GetItemStatus returns the status of *row*. If there is no DataWindow object assigned to the DataWindow control or DataStore, GetItemStatus returns NULL. If any argument value is NULL, the method returns NULL.

Usage

Use GetItemStatus to understand what SQL statements will be generated for new and changed information when you update the database.

For rows in the primary and filter buffers, Update generates an INSERT statement for rows with NewModified! status. It generates an UPDATE statement for rows with DataModified! status and references the columns that have been affected.

For rows in the delete buffer, Update does not generate a DELETE statement for rows whose status was New! or NewModified! before being moved to the delete buffer.

Examples

These statements store in the variable l_status the status of the column named emp_status in row 5 in the filter buffer of dw_1:

```
dwItemStatus l_status
l_status = dw_1.GetItemStatus(5, "emp_status", &
       Filter!)
```

These statements store in the variable l_status the status of the column named Salary in the current row in the primary buffer of dw_emp:

```
dwItemStatus l_status
l_status = dw_emp.GetItemStatus( &
       dw_emp.GetRow(), "Salary", Primary!)
```

See also

SetItemStatus

GetItemString

Description

Gets data whose type is String from the specified buffer of a DataWindow control or DataStore object. You can obtain the data that was originally retrieved and stored in the database from the original buffer, as well as the current value in the primary, delete, or filter buffers.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

string *dwcontrol*.**GetItemString** (long *row*, integer *column* {, DWBuffer *dwbuffer*, boolean *originalvalue* })

string *dwcontrol*.**GetItemString** (long *row*, string *column* {, DWBuffer *dwbuffer*, boolean *originalvalue* })

Argument	Description
dwcontrol	A reference to a DataWindow control, DataStore, or child DataWindow.
row	A value identifying the row location of the string data.
column	The column location of the data. The datatype of the column must be String. *Column* can be a column number or a column name. The column number is the number of the column as it is listed in the Column Specification view of the DataWindow painter—not necessarily the number of the column in the Design view. To get the contents of a computed field, specify the name of the computed field for *column*. Computed fields do not have numbers.
dwbuffer (optional)	A value of the dwBuffer enumerated datatype identifying the DataWindow buffer from which you want to get the data. For a list of valid values, see DWBuffer on page 372.
originalvalue (optional)	A boolean indicating whether you want the original or current values for *row* and *column*: • True — Returns the original values (the values initially retrieved from the database). • False — (Default) Returns the current values. If you specify *dwbuffer*, you must also specify *originalvalue*.

Return value

Returns the string value in the specified row and column. Returns the empty string ("") if there is no DataWindow object assigned to the DataWindow control or DataStore or if any other error occurs.

If any argument value is NULL, the method returns NULL.

Usage

Use GetItemString to get information from the DataWindow's buffers. To find out what the user entered in the current column before that data is accepted, use GetText. In the ItemChanged or ItemError events, use the data argument.

To access a row in the original buffer, specify the buffer that the row currently occupies (primary, delete, or filter) and the number of the row in that buffer. When you specify TRUE for *originalvalue*, the method gets the original data for that row from the original buffer.

GetItemString returns a formatted value in the case of a computed column, and an unformatted value in the case of a noncomputed column. In PowerBuilder, you can use the GetItemFormattedString method to return a formatted value, or the GetItemUnformattedString method to return an unformatted value, for any type of column.

Mismatched datatypes
An execution error occurs when the datatype of the DataWindow column does not match the datatype of the method—in this case, String.

Examples

These statements set LName to the current string in the primary buffer for row 3 of the column named emp_name in the DataWindow dw_employee:

```
String LName
LName = dw_employee.GetItemString(3, "emp_name")
```

These statements set LName to the current string in the delete buffer for row 3 of the column named emp_name of dw_employee:

```
String LName
LName = dw_employee.GetItemString(3, &
       "emp_name", Delete!, FALSE)
```

The following statements set LName to the original string in the delete buffer for row 3 of the column named emp_name of dw_employee:

```
String LName
LName = dw_employee.GetItemString(3, &
       "emp_name", Delete!, TRUE)
```

See also

GetItemDate
GetItemDateTime
GetItemDecimal
GetItemFormattedString
GetItemNumber
GetItemTime
GetItemUnformattedString
GetText
SetItem
SetText

GetItemTime

Description

Gets data whose type is Time from the specified buffer of a DataWindow control or DataStore object. You can obtain the data that was originally retrieved and stored in the database from the original buffer, as well as the current value in the primary, delete, or filter buffers.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

time *dwcontrol*.**GetItemTime** (long *row*, string *column* {, DWBuffer *dwbuffer*, boolean *originalvalue* })

time *dwcontrol*.**GetItemTime** (long *row*, integer *column* {, DWBuffer *dwbuffer*, boolean *originalvalue* })

Argument	Description
dwcontrol	A reference to a DataWindow control, DataStore, or child DataWindow.
row	A value identifying the row location of the data.
column	The column location of the data. The datatype of the column must be time. *Column* can be a column number or a column name. The column number is the number of the column as it is listed in the Column Specification view of the DataWindow painter—not necessarily the number of the column in the Design view. To get the contents of a computed field, specify the name of the computed field for *column*. Computed fields do not have numbers.
dwbuffer (optional)	A value of the dwBuffer enumerated datatype identifying the DataWindow buffer from which you want to get the data. For a list of valid values, see DWBuffer on page 372.
originalvalue (optional)	A boolean indicating whether you want the original or current values for *row* and *column*: • True — Return the original values (the values initially retrieved from the database). • False — (Default) Return the current values. If you specify *dwbuffer*, you must also specify *originalvalue*.

Return value

Returns the time value in the specified row and column. Returns NULL if the column value is NULL or if there is no DataWindow object assigned to the DataWindow control or DataStore. Returns 00:00:00.000000 if an error occurs. If any argument value is NULL, the method returns NULL.

Usage

Use GetItemTime to get information from the DataWindow's buffers. To find out what the user entered in the current column before that data is accepted, use GetText. In the ItemChanged or ItemError events, use the data argument.

To access a row in the original buffer, specify the buffer that the row currently occupies (primary, delete, or filter) and the number of the row in that buffer. When you specify TRUE for *originalvalue*, the method gets the original data for that row from the original buffer.

Datatypes of columns and computed fields
An execution error occurs when the datatype of the DataWindow column does not match the datatype of the method—in this case, time.

There is a difference in datatypes between computed columns retrieved from the database and computed fields defined in the DataWindow painter. Computed columns from the database can have a datatype of time, but a time computed field always has a datatype of DateTime, not time. Use the GetItemDateTime method instead.

Using GetItemTime in a String function
When you call GetItemTime 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 "2/26/96 00:00:00":

```
String(dw_1.GetItemTime(1, "start_date"))
```

To get a simple time string, you can specify a display format for the String function or you can assign the value to a time variable before calling the String function (see GetItemDate for examples).

Examples

These statements set Start to the current Time data in the primary buffer for row 3 of the column named title in dw_employee:

```
Time Start
Start = dw_employee.GetItemTime(3, "title")
```

These statements set Start to the current Time data in the filter buffer for row 3 of the column named start_time of dw_employee:

```
Time Start
Start = dw_employee.GetItemTime(3, &
       "start_time", Filter!, FALSE)
```

These statements set Start to the original Time data in the primary buffer for row 3 of the column named start_time of dw_employee:

```
Time Start
Start = dw_employee.GetItemTime(3, &
       "start_time", Primary!, TRUE)
```

See also

GetItemDate
GetItemDateTime
GetItemDecimal
GetItemNumber
GetItemString
GetText
SetItem
SetText

GetItemUnformattedString

Description

Gets raw (unformatted) data whose type is String from the specified buffer of a DataWindow control or DataStore object.

PocketBuilder	✗
PowerBuilder	✓

Syntax

PowerBuilder DataWindow control, DataStore object, or child DataWindow

string *dwcontrol*.**GetItemUnformattedString** (long *row*, integer *column* {, DWBuffer *dwbuffer*, boolean *originalvalue* })

string *dwcontrol*.**GetItemUnformattedString** (long *row*, string *column* {, DWBuffer *dwbuffer*, boolean *originalvalue* })

Return value

String. Returns the value of the data without its display formatting.

GetLastError

Description

Returns the error code of the last database error that occurred in the Web DataWindow server component.

PocketBuilder	✗
PowerBuilder	✓

Syntax **Web DataWindow server component**

long *dwcontrol*.**GetLastError** ( )

Return value Returns a numeric error code for the last database error that occurred. If *dwcontrol* is NULL, the method returns NULL.

GetLastErrorString

Description Returns the text of the error message for the last database error that occurred in the Web DataWindow server component.

PocketBuilder	✗
PowerBuilder	✓

Syntax **Web DataWindow server component**

string *dwcontrol*.**GetLastErrorString** ( )

Return value Returns a string containing an error message for the last database error that occurred. If *dwcontrol* is NULL, the method returns NULL.

GetMessageText

Description Obtains the message text generated by a crosstab DataWindow object in a DataWindow control. Only crosstab DataWindows generate messages.

PocketBuilder	✗
PowerBuilder	✓

Obsolete method
GetMessageText is obsolete. You should replace all use of GetMessageText as soon as possible. The message text is available as an argument in a user event defined for pbm_dwnmessagetext in a DataWindow control.

Syntax **PowerBuilder DataWindow control**

string *dwcontrol*.**GetMessageText** ( )

Return value

Returns the text of the message generated by *dwcontrol*. If there is no text or an error occurs, GetMessageText returns the empty string (""). If *dwcontrol* is NULL, the method returns NULL.

GetNextModified

Description

Reports the next row that has been modified in the specified buffer.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

long *dwcontrol*.**GetNextModified** (long *row*, DWBuffer *dwbuffer*)

Argument	Description
dwcontrol	A name of the DataWindow control, DataStore, or child DataWindow in which you want to locate the modified row.
row	A value identifying the row location after which you want to locate the modified row. To search from the beginning, specify 0.
dwbuffer	A value of the dwBuffer enumerated datatype identifying the DataWindow buffer in which you want to locate the modified row. For a list of valid values, see DWBuffer on page 372.

Return value

Returns the number of the first row that was modified after *row* in *dwbuffer* in *dwcontrol*. Returns 0 if there are no modified rows after the specified row. If any argument value is NULL, the method returns NULL.

Usage

PocketBuilder stores the update status of rows and columns in the DataWindow. The status settings indicate whether a row or column is new or has been modified. GetNextModified reports rows with the status NewModified! and DataModified!.

For more information on the status of rows and columns, see GetItemStatus and SetItemStatus.

Using GetNextModified on the delete buffer will return rows that have been modified and then deleted. The DeletedCount method will report the total number of deleted rows.

GetNextModified begins searching in the row after the value you specify in *row*. This is different from the behavior of Find, FindGroupChange, and FindRequired, which begin searching in the row you specify.

Total number of modified rows
You can use the ModifiedCount method to find out the total number of modified rows in the primary and filter buffers.

Examples

These statements count the number or rows that were modified in the primary buffer for dw_status and then display a message reporting the number modified:

```
integer rc
long NbrRows, ll_row = 0, count = 0

dw_status.AcceptText()
NbrRows = dw_status.RowCount()
DO WHILE ll_row <= NbrRows
       ll_row = dw_status.GetNextModified(ll_row,&
          Primary!)
       IF ll_row > 0 THEN
          count = count + 1
       ELSE
          ll_row = NbrRows + 1
       END IF
LOOP
MessageBox("Modified Count", &
       String(count) &
       + " rows were modified.")
```

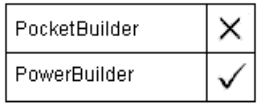
See also

DeletedCount
FindRequired
GetNextModified
ModifiedCount
SetItemStatus

GetObjectAtPointer

Description

Reports the control within the DataWindow object and row number under the pointer. Controls include columns, labels, and other graphic controls, such as lines and pictures.



Syntax

PowerBuilder DataWindow control

string *dwcontrol*.**GetObjectAtPointer** ( )

Return value

Returns the string whose value is the name of the control under the pointer, followed by a tab character and the row number. Returns the empty string ("") if an error occurs. If *dwcontrol* is NULL, the method returns NULL.

GetParent

Description

Obtains the parent of the specified object.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

PowerObject *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

Returns a reference to the parent of *objectname*.

Usage

Inherited from PowerObject. For information, see GetParent in the *PowerScript Reference*.

GetRow

Description

Reports the number of the current row in a DataWindow control or DataStore object.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

long *dwcontrol*.**GetRow** ( )

Argument	Description
dwcontrol	A reference to a DataWindow control, DataStore, or the child DataWindow

Return value

Returns the number of the current row in *dwcontrol*. Returns 0 if no row is current and -1 if an error occurs. If *dwcontrol* is NULL, the method returns NULL.

Current row not always displayed
The current row is not always a row displayed on the screen. For example, if the cursor is on row 7 column 2 and the user uses the scroll bar to scroll to row 50, the current row remains row 7 unless the user clicks row 50.

Examples

This statement returns the number of the current row in dw_Employee:

```
dw_employee.GetRow()
```

See also

GetColumn
SetColumn
SetRow

GetRowFromRowId

Description

Gets the row number of a row in a DataWindow control or DataStore object from the unique row identifier associated with that row.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

long *dwcontrol*.**GetRowFromRowId** (long *rowid* {, DWBuffer *buffer* })

Argument	Description
dwcontrol	A reference to a DataWindow control, DataStore, or child DataWindow.
rowid	A number specifying the row identifier for which you want the associated row number.

Argument	Description
buffer (optional)	A value of the dwBuffer enumerated datatype identifying the DataWindow buffer that contains the row. For a list of valid values, see DWBuffer on page 372.

Return value

Returns the row number in *buffer*. Returns 0 if the row number is not in the current buffer and -1 if an error occurs. If any argument value is NULL, the method returns NULL.

Usage

This method allows you to use a unique row identifier to retrieve the associated DataWindow or DataStore row number. The row identifier is not affected by operations (such as Insert, Delete, or Filter) that might change the original order (and consequently the row numbers) of the rows in the DataWindow or DataStore.

Row identifiers
The row identifier is relative to the DataWindow that currently owns the row.

Examples

This example uses the row identifier previously obtained using the GetRowIdFromRow method to retrieve the row's number after the original order of the rows in the DataWindow has changed.

```
long ll_rowid
long ll_rownumber

ll_rowid = dw_1.GetRowIdFromRow(dw_1.GetRow())
// suppose original order of rows changes...
ll_rownumber = dw_1.GetRowFromRowId(ll_rowid)
```

See also

GetRow
GetRowIdFromRow

GetRowIdFromRow

Description

Gets the unique row identifier of a row in a DataWindow control or DataStore object from the row number associated with that row.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

long *dwcontrol*.**GetRowIdFromRow** (long *rownumber* {, DWBuffer *buffer* })

Argument	Description
dwcontrol	A reference to a DataWindow control, DataStore, or the child DataWindow.
rownumber	A number specifying the row number for which you want the associated row identifier.
buffer (optional)	A value of the dwBuffer enumerated datatype identifying the DataWindow buffer that contains the row. For a list of valid values, see DWBuffer on page 372.

Return value

Returns the row identifier in *buffer*. Returns 0 if the row identifier is not in the current buffer and -1 if an error occurs. If any argument value is NULL, the method returns NULL.

Usage

The row identifier value is not the same as the row number value used in many DataWindow and DataStore function calls and should not be used for the row number value. Instead you should first convert the unique row identifier into a row number by calling GetRowFromRowId.

Row identifiers
The row identifier is relative to the DataWindow that currently owns the row.

Examples

This example retrieves the current row's unique identifier:

```
long ll_rowid
ll_rowid = dw_emp.GetRowIDFromRow(dw_emp.GetRow())
```

See also

GetRow
GetRowFromRowId

GetSelectedRow

Description

Reports the number of the next highlighted row after a specified row in a DataWindow control or DataStore object.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

long *dwcontrol*.**GetSelectedRow** (long *row*)

Argument	Description
dwcontrol	A reference to a DataWindow control, DataStore, or child DataWindow.
row	A value identifying the location of the row after which you want to search for the next selected row. Specify 0 to begin searching at the first row.

Return value

Returns the number of the first row that is selected after *row* in *dwcontrol*. Returns 0 if no row is selected after the specified row. If any argument value is NULL, the method returns NULL.

Usage

Rows are not automatically selected—that is, highlighted—when they become current. You can select a row by calling the SelectRow method.

GetSelectedRow begins its search *after* the specified row. It does not matter whether *row* itself is selected.

Examples

This statement returns the number of the first row that is selected in dw_Employee:

```
dw_employee.GetSelectedRow(0)
```

This statement returns the number of the first row that is selected beginning with row 25 in dw_Employee:

```
dw_employee.GetSelectedRow(25)
```

See also

SelectRow

GetSQLPreview

Description

Reports the SQL statement that the DataWindow control is currently submitting to the database.

PocketBuilder	✗
PowerBuilder	✓

Obsolete method
GetSQLPreview is obsolete. You should replace all references to GetSQLPreview as soon as possible. The SQL syntax is available as an argument in the DBError and SQLPreview events.

Syntax **PowerBuilder DataWindow control or child DataWindow**

string *dwcontrol*.**GetSQLPreview** ( )

Return value Returns the current SQL statement for *dwcontrol*. Returns the empty string ("") if an error occurs. If *dwcontrol* is NULL, the method returns NULL.

GetSQLSelect

Description Reports the SQL SELECT statement associated with a DataWindow if its data source is one that accesses a SQL database (such as SQL Select, Quick Select, or Query).

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax string *dwcontrol*.**GetSQLSelect** ( )

Argument	Description
dwcontrol	A reference to a DataWindow control, DataStore, or child DataWindow.

Return value Returns the current SQL SELECT statement for *dwcontrol*. GetSQLSelect returns the empty string ("") if it cannot return the statement. If *dwcontrol* is NULL, the method returns NULL.

Usage When you want to change the SQL SELECT statement for a DataWindow or DataStore during execution, you can use GetSQLSelect to save the current SELECT statement before making the change.

When you define a DataWindow, PocketBuilder stores a PocketBuilder SELECT statement (PBSELECT) with the DataWindow. If a database is connected and SetTransObject has been called for the DataWindow, then GetSQLSelect returns the SQL SELECT statement. Otherwise, GetSQLSelect returns the PBSELECT statement.

You can also use Describe to obtain the SQL SELECT statement. The DataWindow object's Table.Select property holds the information.

Examples

The code saves the SELECT statement for dw_emp in the variable old_select. Then it adds a WHERE clause. The example assumes the old SELECT statement did not have one already:

```
string old_select, new_select, where_clause
// Get old SELECT statement
old_select = dw_emp.GetSQLSelect()

// Specify new WHERE clause
where_clause = "WHERE ..."
// Add the new where clause to old_select
new_select = old_select + where_clause

// Set the SELECT statement for the DW
dw_emp.SetSQLSelect(new_select)
```

See also

SetSQLSelect

GetStateStatus

Description

Retrieves the current status of the internal state flags for a DataWindow and places this information in a blob. This method is used primarily in distributed applications.

PocketBuilder	✗
PowerBuilder	✓

Obsolete method
GetStateStatus is obsolete. This method was originally added to PowerScript to allow you to synchronize a source DataWindow with multiple target DataWindows. This technique is no longer supported.

Syntax

PowerBuilder DataWindow control or DataStore object

long *dwcontrol*.**GetStateStatus** (blob *cookie*)

Return value

Returns 1 if it succeeds and -1 if it fails. If any argument value is NULL, the method returns NULL.

GetText

Description

Obtains the value in the edit control over the current row and column. When the user changes a value in a DataWindow, it is available in the edit control before it is accepted into the column.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

string *dwcontrol*.**GetText** ( )

Argument	Description
dwcontrol	A reference to a DataWindow control, DataStore, or child DataWindow

Return value

Returns the value in the edit control over the current row and column in *dwcontrol*. The value might or might not have been accepted into the row and column. Returns the empty string ("") if no column is currently selected in *dwcontrol*. If *dwcontrol* is NULL, the method returns NULL.

Usage

The values in the rows and columns of a DataWindow are items in the DataWindow's buffer. When a user edits a value in a row and column, the item value is transferred as text to the edit control, where the user can change the value. When the user leaves the column or when a script calls AcceptText, the text in the edit control is accepted into the column and becomes the value of the item in the buffer.

You do not need to call GetText in the script for the ItemChanged or ItemError event. To check the value entered in the edit control over the current row and column before allowing it to be accepted into the column, use the data argument.

To obtain the value stored in the DataWindow's buffer for the row and column, use the GetItem method that corresponds with the datatype of the column.

Examples

These statements return the text in the edit control of dw_employee:

```
string LName
LName = dw_employee.GetText()
```

See also

SetText

GetTrans

Description

Gets the values for the DataWindow control or DataStore object's internal transaction object and stores these values in the programmer-specified transaction object.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

integer *dwcontrol*.**GetTrans** (transaction *transaction*)

Argument	Description
dwcontrol	A reference to a DataWindow control, DataStore, or child DataWindow
transaction	The name of the transaction object into which you want to put the values

Return value

Returns 1 if it succeeds and -1 if an error occurs. The return value is usually not used. If any argument value is NULL, the method returns NULL.

Usage

The SetTrans method (not the SetTransObject method) sets the internal transaction object. If you have not called SetTrans, GetTrans will fail.

Use GetTrans when you want to get the values for the transaction object in order to modify them, as shown in the last example.

If you are using SetTransObject, which specifies transaction information using a programmer-specified transaction object, GetTrans will not report information about the programmer-specified transaction object currently in effect. (SetTransObject is the recommended connection method because it gives better application performance. See SetTrans and SetTransObject for more information.)

Examples

This example puts the values in the internal transaction object for dw_employee into the programmer-specified transaction object named object1:

```
transaction object1
object1 = CREATE transaction
dw_employee.GetTrans(object1)
```

The following statement puts the values in the internal transaction object for dw_employee into the default transaction object (SQLCA):

```
dw_employee.GetTrans(SQLCA)
```

The following statements change the database type and password of dw_employee. The first two statements create the transaction object emp_TransObj. The next two statements use the SetTrans method to set the values of SQLCA, and then use the GetTrans method to store the values of the current transaction object for dw_employee in emp_TransObj. The last two statements change the database type and password, and then the SetTrans method puts the revised values in the transaction object for dw_employee:

```
// Name the transaction object.
transaction emp_TransObj

// Create the transaction object.
emp_TransObj = CREATE transaction

// Set the internal transaction object.
dw_employee.SetTrans(SQLCA)

// Fill the new transaction object with original
// values from SQLCA.
dw_employee.GetTrans(emp_TransObj)

// Put revised values into the new transaction
// object.
// Change the database type.
emp_TransObj.DBMS = "Sybase"

// Change the password.
emp_TransObj.LogPass = "cam2"

// Associate the new transaction object with
// dw_employee, replacing SQLCA.
dw_employee.SetTrans(emp_TransObj)
```

See also SetTrans

GetUpdateStatus

Description

Reports the row number and buffer of the row that is currently being updated in the database. When called because of an error, GetUpdateStatus reports the row that caused the error.

PocketBuilder	✗
PowerBuilder	✓

Obsolete method
GetUpdateStatus is obsolete. The update status is available as an argument in the DBError and SQLPreview events.

Syntax

PowerBuilder DataWindow control or child DataWindow

integer *dwcontrol*.**GetUpdateStatus** (long *row*, DWBuffer *dwbuffer*)

Return value

Returns 1 if it succeeds and -1 if an error occurs. The number and buffer of the row currently being updated are stored in *row* and *dwbuffer*. If any argument value is NULL, the method returns NULL.

GetValidate

Description

Obtains the validation rule for a column in a DataWindow.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

string *dwcontrol*.**GetValidate** (string *column*)

string *dwcontrol*.**GetValidate** (integer *column*)

Argument	Description
dwcontrol	A reference to a DataWindow control, DataStore, or child DataWindow.
column	The column for which you want the validation rule. *Column* can be a column number or a column name. The column number is the number of the column as it is listed in the Column Specification view of the DataWindow painter—not necessarily the number of the column in the Design view.

Return value

Returns the validation rule for *column* in *dwcontrol*. Returns the empty string ("") if no validation criteria are defined for the column. If any argument value is NULL, the method returns NULL.

Usage

You can use GetValidate to save the current validation rule before calling SetValidate to change the rule temporarily.

Examples

These statements change the validation rule for column 7 in the DataWindow control dw_Employee to Rule2:

```
string Rule1, Rule2 = "Long(GetText()) > 15000"
Rule1 = dw_Employee.GetValidate(7)
dw_Employee.SetValidate(7, Rule2)
```

See also

SetValidate

GetValue

Description

Obtains the value of an item in a value list or code table associated with a column in a DataWindow.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

string *dwcontrol*.**GetValue** (string *column*, integer *index*)

string *dwcontrol*.**GetValue** (integer *column*, integer *index*)

Argument	Description
dwcontrol	A reference to a DataWindow control, DataStore, or child DataWindow.
column	The column for which you want the item. *Column* can be a column number (integer) or a column name (string).
index	The number of the item in the value list or the code table for the edit style.

Return value

Returns the item identified by *index* in the value list or the code table associated with *column* of *dwcontrol*. If the item has a display value that is not the actual value, GetValue returns a tab-separated string consisting of:

displayvalue[tab]*codevalue*

Returns the empty string ("") if the index is not valid or the column does not have a value list or code table. If any argument value is NULL, the method returns NULL.

Usage

You can use GetValue to find out the values associated with the following edit styles: CheckBox, RadioButton, DropDownListBox, Edit Mask, and Edit. If the edit style has a code table in which each value in the list has a display value and a data value, GetValue reports both values.

GetValue does not get values from a DropDownDataWindow code table.

You can parse the return value by searching for the tab character (ASCII 09). In PocketBuilder, search for ~t.

Examples

If the value list for column 7 of dw_employee contains Full Time, Part Time, Retired, and Terminated, these statements return the value of item 3 (Retired):

```
string Status
Status = dw_employee.GetValue(7,3)
```

If the value list for the column named product of dw_employee is Widget[tab]1, Gadget[tab]2, the following code returns Gadget[tab]2 and saves the display value in a string variable:

```
string ls_prodinfo, ls_prodname, ls_prodnum
integer li_tab

ls_prodinfo = dw_employee.GetValue("product", 2)

li_tab = Pos(ls_prodinfo, "~t", 1)
ls_prodname = Left(ls_prodinfo, li_tab - 1)
ls_prodnum = Mid(ls_prodinfo, li_tab + 1)
```

See also

ClearValues
SetValue

GroupCalc

Description

Recalculates the breaks in the grouping levels in a DataWindow.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

integer *dwcontrol*.**GroupCalc** ( )

Argument	Description
dwcontrol	A reference to a DataWindow control, DataStore, or child DataWindow

Return value

Returns 1 if it succeeds and -1 if an error occurs. If *dwcontrol* is NULL, the method returns NULL.

Usage

Use GroupCalc to force the DataWindow object to recalculate the breaks in the grouping levels after you have added or modified rows in a DataWindow.

GroupCalc does not sort the data before it recalculates the breaks. Therefore, unless you populated the DataWindow in a sorted order, call the Sort method to sort the data before you call GroupCalc.

Examples

This code imports new rows from a file into the DataWindow dw_emp and then recalculates the group breaks for dw_emp:

```
dw_emp.ImportFile("d:\employee.txt")
dw_emp.SetRedraw(false)
dw_emp.SetSort("1A")
dw_emp.Sort()
dw_emp.GroupCalc()
dw_emp.SetRedraw(true)
```

See also

Sort

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	✓

Syntax

Integer *objectname*.**Hide** ( )

Argument	Description
objectname	The name of the object or control you want to make invisible

Return value

Returns 1 if it succeeds and -1 if an error occurs. If *objectname* is NULL, Hide returns NULL.

Usage

Inherited from GraphicObject. For information, see Hide in the *PowerScript Reference*.

ImportClipboard

Description

Inserts data into a DataWindow control or DataStore object 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.

Syntax

long *dwcontrol*.**ImportClipboard** ({saveastype *importtype*}, { long *startrow* {, long *endrow* {, long *startcolumn* {, long *endcolumn* {, long *dwstartcolumn* } } } } })

Argument	Description
importtype (optional)	An enumerated value of the SaveAsType DataWindow constant or a number representing that value (see SaveAsType on page 378). Valid import type arguments for ImportClipboard are: Text! CSV! XML! (PowerBuilder only)
dwcontrol	A reference to a DataWindow control, DataStore, or child DataWindow.
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.
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.

Argument	Description
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.
endcolumn (optional)	The number of the last column in the clipboard that you want to copy. The default is the rest of the columns. For default XML import, if *endcolumn* is supplied and is smaller than *N*, where *N* is the DataWindow row size, import skips the last (*N* - *endcolumn*) elements in each row. This argument has no effect on template XML import.
dwstartcolumn (optional)	The number of the first column in the DataWindow control or DataStore that should receive data. The default is 1.

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

-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

-13 Unsupported DataWindow style for import

-14 Error resolving DataWindow nesting

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.

If an XML or CSV column contains a leading double quote, it is assumed to be part of the column value. A leading double quote has to be closed to mark the end of an item.

All the arguments of this function are optional. You do not need to specify the *importtype* argument. The *startcolumn* and *endcolumn* arguments control the number of imported columns and the number of columns in the DataWindow that are affected. The *dwstartcolumn* argument specifies the first DataWindow column to be affected. The following formula calculates the last column to be affected.

dwstartcolumn + (*endcolumn* - *startcolumn*)

ImportClipboard does not support Crosstab DataWindow objects.

Examples

This statement copies all data in the clipboard to the DataWindow dw_employee starting at the first column:

```
dw_employee.ImportClipboard()
```

This statement copies all data in the clipboard to the DataWindow dw_employee starting at the first column and specifies that the data is in XML format:

```
dw_employee.ImportClipboard()
```

This statement inserts data from the clipboard into the DataWindow dw_employee. It copies rows 2 through 30 and columns 3 through 8 on the clipboard to the DataWindow beginning in column 5. It adds 29 rows to the DataWindow with data in columns 5 through 10:

```
dw_employee.ImportClipboard(2,30,3,8,5)
```

See also

ImportFile
ImportString

ImportFile

Description

Inserts data into a DataWindow control or DataStore from a file. The data can be tab-separated text, comma-separated text, or XML.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

XML data
XML data is not supported in this release of PocketBuilder.

Syntax

long *dwcontrol*.**ImportFile** ({saveastype *importtype*}, string *filename* {, long *startrow* {, long *endrow* {, long *startcolumn* {, long *endcolumn* {, long *dwstartcolumn* } } } } })

Argument	Description
dwcontrol	A reference to a DataWindow control or DataStore.
importtype (optional)	An enumerated value of the SaveAsType DataWindow constant or a number representing that value (see SaveAsType on page 378). If this argument is specified, the *filename* 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 XML. Specify the file's full name. If the optional *importtype* is not specified, the name must end in the appropriate extension. If *filename* is an empty string, ImportFile displays the File Open dialog box and allows the user to select a file. The remaining arguments are ignored.
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.

Argument	Description
endcolumn (optional)	The number of the last column in the file that you want to copy. The default is the rest of the columns. For default XML import, if *endcolumn* is supplied and is smaller than *N*, where *N* is the DataWindow row size, import skips the last (*N* - *endcolumn*) elements in each row. This argument has no effect on template XML import.
dwstartcolumn (optional)	The number of the first column in the DataWindow control or DataStore that should receive data. The default is 1.

Events

ImportFile can trigger an ItemError event.

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

-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 or *.csv)

-13 Unsupported DataWindow style for import

-14 Error resolving DataWindow nesting

Usage

In an application running on a Pocket PC or Smartphone, you can import Unicode files only. The file should consist of rows of data. If the file includes column headings or row labels, set the *startrow* and *startcolumn* arguments to skip them. The datatypes and order of the DataWindow object's columns must match the columns of data in the file.

The *startcolumn* and *endcolumn* arguments control the number of columns imported from the file and the number of columns in the DataWindow that are affected. The *dwstartcolumn* argument specifies the first DataWindow column to be affected. The following formula calculates the last DataWindow to be affected.

dwstartcolumn + (*endcolumn* - *startcolumn*)

To let users select the file to import, specify a NULL string for *filename*. PocketBuilder displays the Select Import File dialog box. A drop-down list lets the user select the type of file to import.

Specifying a NULL string for filename
If you specify a NULL string for *filename*, the remaining arguments are ignored. All the rows and columns in the file are imported.

Double quotes The location and number of double quote marks in a field in a tab-separated 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 or comma 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-separated strings are imported into a two-column DataWindow:

Table 9-4: Examples of strings 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
"Mich"ael TAB """Mariam"""	Mich "Mariam"
"Amy TAB Doherty"	Amy<TAB>Doherty in first cell, second cell empty
3""" TAB 4"	3""" 4"

If an XML or CSV column contains a leading double quote, it is assumed to be part of the column value. A leading double quote has to be closed to mark the end of an item.

Examples

This statement inserts all the data in the file D:\TMP\EMPLOYEE.CSV into dw_employee starting at the first column:

```
dw_employee.ImportFile("D:\TMP\EMPLOYEE.CSV")
```

This statement inserts all the data in the file D:\TMP\EMPLOYEE.XML into dw_employee starting at the first column:

```
dw_employee.ImportFile(XML!,"D:\TMP\EMPLOYEE")
```

The following statements are equivalent. Both import the contents of the XML file named *myxmldata*:

```
dw_control.ImportFile(myxmldata.xml)
dw_control.ImportFile(XML!, myxmldata)
```

This statement inserts the data from the file D:\TMP\EMPLOYEE.TXT into the DataWindow dw_employee. It copies rows 2 through 30 and columns 3 through 8 in the file to the DataWindow beginning in column 5. The result is 29 rows added to the DataWindow with data in columns 5 through 10:

```
dw_employee.ImportFile("D:\TMP\EMPLOYEE.TXT", &
   2, 30, 3, 8, 5)
```

See also

ImportClipboard
ImportString

ImportString

Description

Inserts data into a DataWindow control or DataStore from tab-separated, comma-separated, or XML data in a string.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

XML data
XML data is not supported in this release of PocketBuilder.

Syntax

long *dwcontrol*.**ImportString** ({saveastype *importtype*}, string *string* {, long *startrow* {, long *endrow* {,long *startcolumn* {, long *endcolumn* {, long *dwstartcolumn* } } } } })

Argument	Description
dwcontrol	A reference to a DataWindow control or DataStore.
importtype (optional)	An enumerated value of the SaveAsType DataWindow constant or a string or number representing that value (see SaveAsType on page 378). If no import type is specified, the imported string should contain only tab-separated text. Valid type arguments are: Text! (default) CSV! XML! (PowerBuilder only)
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).
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.
endcolumn (optional)	The number of the last column in the string that you want to copy. The default is the rest of the columns. For default XML import, if *endcolumn* is supplied and is smaller than *N*, where *N* is the DataWindow row size, import skips the last (*N* - *endcolumn*) elements in each row. This argument has no effect on template XML import.
dwstartcolumn (optional)	The number of the first column in the DataWindow control or DataStore that should receive data. The default is 1.

Events

ImportString may trigger an ItemError event.

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

-3 Invalid argument

-4 Invalid input

-13 Unsupported DataWindow style for import

-14 Error resolving DataWindow nesting

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

All the arguments of this function except *string* are optional. You do not need to specify the *importtype* argument.

The string must be formatted in tab-separated or comma-separated columns or in XML. For TXT and CSV files, the format of the string is the same as if the data came from an ASCII file, and each line must end with a carriage return and a newline character (~r~n). If the string has four tab-separated columns, one line might look like for a tab-separated string:

col1_data~t *col2_data*~t *col3_data*~t *col4_data*~r~n

For a DataWindow control or DataStore, the string should consist of rows of data. If the data includes column headings or row labels, set the *startrow* and *startcolumn* arguments to skip them. The datatypes and order of the DataWindow object's columns must match the columns of data in the string.

The *startcolumn* and *endcolumn* arguments control the number of columns imported from the string and the number of columns in the DataWindow that are affected. The *dwstartcolumn* argument specifies the first DataWindow column to be affected. The following formula calculates the last DataWindow to be affected.

dwstartcolumn + (*endcolumn* - *startcolumn*)

If string data to be assigned to a single row and column has multiple lines (indicated by line-ending characters in the import string), you must quote the string data using ~". Do not use single quotes.

This example of a valid tab-separated import string assigns multiline values to each row in column 2:

```
ls_s = &
       "1~t~"Mickey~r~nMinnie~r~nGoofy~" ~r~n" + &
       "2~t~"Susan~r~nMary~r~nMarie~" ~r~n" + &
       "3~t~"Chris~r~nBen~r~nMike~" ~r~n" + &
       "4~t~"Mott~r~nBarber~r~nPicard~" "
```

If an XML or CSV column contains a leading double quote, it is assumed to be part of the column value. A leading double quote has to be closed to mark the end of an item.

ImportString does not support Crosstab DataWindow objects.

Examples

These statements copy all data in the string ls_Emp_Data to the DataWindow control dw_employee starting at the first column:

```
string ls_Emp_Data
ls_Emp_Data = . . .
dw_employee.ImportString(ls_Emp_Data)
```

This statement stores data in the string ls_Text and imports it into the DataWindow dw_employee. The DataWindow is a report of department 100 and start and end dates of personnel. The string includes the department number and other information, which is not imported. ImportString imports rows 2 through 10 and columns 2 through 5 in the string to the DataWindow beginning in column 2. The result is 9 rows added to the DataWindow with data in columns 5 through 8:

```
string ls_text
ls_text = "Dept~tLName~tFName~tStart" &
       + "~tEnd~tAmount~tOutcome ~r~n"
ls_text = ls_text + &
       "100~tJones~tMary~tApr88~tJul94~t40~tG~r~n"
ls_text = ls_text + &
       "100~tMarsh~tMarsha~tApr89~tJan92~t35~tG~r~n"
ls_text = ls_text + &
       "100~tJames~tHarry~tAug88~tMar93~t22~tM~r~n"
...
ls_text = ls_text + &
       "100~tWorth~tFrank~tSep87~tJun94~t55~tE~r~n"

dw_employee.ImportString(ls_text, 2, 10, 2, 5, 5)
```

See also

ImportClipboard
ImportFile

InsertDocument

Description

Inserts a rich text format or plain text file into a DataWindow control or DataStore object.

PocketBuilder	✗
PowerBuilder	✓

Syntax

PowerBuilder DataWindow control or DataStore object

integer *dwcontrol*.**InsertDocument** (string *filename*, boolean *clearflag* {, FileType *filetype* })

Return value

Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is NULL, InsertDocument returns NULL.

InsertRow

Description

Inserts a row in a DataWindow or DataStore. If any columns have default values, the row is initialized with these values before it is displayed.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

long *dwcontrol*.**InsertRow** (long *row*)

Argument	Description
dwcontrol	A reference to a DataWindow control, DataStore, or child DataWindow.
row	A value identifying the row before which you want to insert a row. To insert a row at the end, specify 0.

Return value

Returns the number of the row that was added if it succeeds and -1 if an error occurs. If any argument's value is NULL, the method returns NULL. If there is no DataWindow object assigned to the DataWindow control or DataStore, this method returns -1.

Usage

InsertRow simply inserts the row without changing the display or the current row. To scroll to the row and make it the current row, call ScrollToRow. To simply make it the current row, call SetRow.

A newly inserted row (with a status flag of New!) is not included in the modified count until data is entered in the row (its status flag becomes NewModified!).

Examples

This statement inserts an initialized row before row 7 in dw_Employee:

```
dw_Employee.InsertRow(7)
```

This example inserts an initialized row after the last row in dw_employee, then scrolls to the row, which makes it current:

```
long ll_newrow
ll_newrow = dw_employee.InsertRow(0)
dw_employee.ScrollToRow(ll_newrow)
```

See also

DeleteRow
Update

IsSelected

Description

Determines whether a row is selected in a DataWindow or DataStore. A selected row is highlighted using reverse video.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

boolean *dwcontrol*.**IsSelected** (long *row*)

Argument	Description
dwcontrol	A reference to a DataWindow control, DataStore, or child DataWindow
row	A value identifying the row you want to test to see if it is selected

Return value

Returns TRUE if *row* in *dwcontrol* is selected and FALSE if it is not selected. If *row* is greater than the number of rows in *dwcontrol* or is 0 or negative, IsSelected also returns FALSE. If any argument's value is NULL, the method returns NULL.

Usage

You can call IsSelected in a script for the Clicked event to determine whether the row the user clicked was selected.

Examples

This code calls IsSelected to test whether the current row in dw_employee is selected. If the row is selected, SelectRow deselects it; if it is not selected, SelectRow selects it:

```
long CurRow
boolean result

CurRow = dw_employee.GetRow()
result = dw_employee.IsSelected(CurRow)

IF result THEN
       dw_employee.SelectRow(CurRow, FALSE)
ELSE
       dw_employee.SelectRow(CurRow, TRUE)
END IF
```

This code uses the NOT operator on the return value of IsSelected to accomplish the same result as the IF/THEN/ELSE statement above:

```
integer CurRow
boolean result
CurRow = dw_employee.GetRow()
dw_employee.SelectRow(CurRow, &
       NOT dw_employee.IsSelected(CurRow))
```

See also

SelectRow

LineCount

Description

Determines the number of lines in an edit control that allows multiple lines.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

long *dwcontrol*.**LineCount** ( )

Argument	Description
dwcontrol	A reference to a DataWindow control

Return value

Returns the number of lines in *dwcontrol* if it succeeds and -1 if an error occurs. If *dwcontrol* 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.

Using with other controls
For use of this method with other PocketBuilder controls, see LineCount in the *PowerScript Reference*.

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
```

ModifiedCount

Description

Reports the number of rows that have been modified but not updated in a DataWindow or DataStore.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

long *dwcontrol*.**ModifiedCount** ( )

Argument	Description
dwcontrol	A reference to a DataWindow control, DataStore, or child DataWindow

Return value

Returns the number of rows that have been modified in the primary buffer. Returns 0 if no rows have been modified or if all modified rows have been updated in the database table. Returns -1 if an error occurs. If *dwcontrol* is NULL, the method returns NULL.

Usage

ModifiedCount reports the number of rows that are scheduled to be added or updated in the database table associated with a DataWindow or DataStore. This includes rows in the primary and filter buffers.

A newly inserted row (with a status flag of New!) is not included in the modified count until data is entered in the row (its status flag becomes NewModified!).

The DeletedCount method counts the number of rows in the deleted buffer. The RowCount method counts the total number of rows in the primary buffer.

Examples

If five rows in dw_Employee have been modified but not updated in the associated database table or filtered out of the primary buffer, the following code sets ll_Rows equal to 5:

```
long ll_Rows
ll_Rows = dw_Employee.ModifiedCount()
```

If any rows in dw_Employee have been modified but not updated in the associated database table, this statement updates the database table associated with the dw_employee DataWindow control:

```
IF dw_employee.ModifiedCount() > 0 THEN &
       dw_employee.Update()
```

See also

DeleteRow
DeletedCount
FilteredCount
Retrieve
RowCount
Update

Modify

Description

Modifies a DataWindow object by applying specifications, given as a list of instructions, that change the DataWindow object's definition.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

string *dwcontrol*.**Modify** (string *modstring*)

Argument	Description
dwcontrol	A reference to a DataWindow control, DataStore, or child DataWindow.
modstring	A string whose value is the specifications for the modification. See Usage for appropriate formats.

Return value

Returns the empty string ("") if it succeeds and an error message if an error occurs. The error message takes the form "Line *n* Column *n* incorrect syntax". The character columns are counted from the beginning of the compiled text of *modstring*. If any argument's value is NULL, the method returns NULL.

Usage

You can change appearance, behavior, and database information for the DataWindow object by changing the values of properties. You can add and remove controls from the DataWindow object by providing specifications for the controls. Modify lets you make many of the same settings in a script that you would make in the DataWindow painter. Typical uses for Modify are:

- Changing colors, text settings, and other appearance settings of controls.
- Changing the update status of different tables in the DataWindow so that you can update more than one table.
- Modifying the WHERE clause of the DataWindow object's SQL SELECT statement.
- Turning on Query mode or Prompt For Criteria so users can specify the data they want.
- Changing the status of Retrieve Only As Needed.
- Changing the data source of the DataWindow object.
- Controlling the Print Preview display.
- Deleting and adding controls (such as lines or bitmaps) in the DataWindow object.

Each of these uses is illustrated in the Examples for this method.

You can use three types of statements in *modstring* to modify a DataWindow object.

Statement type	What it does
CREATE *control* (*settings*)	Adds *control* to the DataWindow object (such as text, computed fields, and bitmaps). *Settings* is a list of properties and values using the format you see in exported DataWindow syntax. To create a control, you must supply enough information to define it. *Control* cannot be an OLE Object control. You cannot add an OLE object to a DataWindow using the Modify method.
DESTROY [COLUMN] *control*	Removes *control* from the DataWindow object. When *control* is a column, specify the keyword COLUMN to remove both the column and the column's data from the buffer.
controlname.property=value	Changes the value of *property* to *value*. Properties control the location, color, size, font, and other settings for *controlname*. When *controlname* is DataWindow, you can also set properties for database access. Depending on the specific property, *value* can be: • A constant. • A quoted constant. • An expression that consists of a default value followed by a valid DataWindow expression that returns the appropriate datatype for the property. Expressions are described below.

Object names The DataWindow painter automatically gives names to all controls. In previous versions, it named only columns and column labels and to describe and modify properties of other controls easily, you had to name them.

Expressions for Modify When you specify an expression for a DataWindow property, the expression has the format:

defaultvalue~t*DataWindowpainterexpression*

Defaultvalue is a value that can be converted to the appropriate datatype for the property. It is followed by a tab (~t).

DataWindowpainterexpression is an expression that can use any DataWindow painter function. The expression must also evaluate to the appropriate datatype for the property. When you are setting a column's property, the expression is evaluated for each row in the DataWindow, which allows you to vary the display based on the data. A typical expression uses the If function:

```
'16777215 ~t If(emp_status=~~'A~~',255,16777215)'
```

To use that expression in a modstring, specify the following (entered as a single line):

```
modstring = "emp_id.Color='16777215 ~t
If(emp_status=~~'A~~',255,16777215)'"
```

Not all properties accept expressions. For details on each property, see Chapter 3, "DataWindow Object Properties."

Quotes and tildes Because Modify's argument is a string, which can include other strings, you need to use special syntax to specify quotation marks. To specify that a quotation mark be used within the string rather than match and end a previously opened quote, you can either specify the other style of quote (single quotes nested with double quotes) or precede the quotation mark with a tilde (~).

For another level of nesting, the string itself must specify ~", so you must include ~~ (which specifies a tilde) followed by ~" (which specifies a quote). For example, another way to type the modstring shown above (entered as a single line) is:

```
modstring = "emp_id.Color=~"16777215 ~t
If(emp_status=~~~"A~~~",255,16777215)~""
```

For more information about quotes and tildes, see the section on standard datatypes in the *PowerScript Reference*.

Building a modstring with variables To use variable data in *modstring*, you can build the string using variables in your program. As you concatenate sections of *modstring*, make sure quotes are included in the string where necessary. For example, the following code builds a modstring similar to the one above, but the default color value and the two color values in the If function are calculated in the script. Notice how the single quotes around the expression are included in the first and last pieces of the string:

```
red_amount = Integer(sle_1.Text)
modstring = "emp_id.Color='" + &
       String(RGB(red_amount, 255, 255)) + &
       "~tIf(emp_status=~~'A~~'," + &
       String(RGB(255, 0, 0)) + &
```

```
        "," + &
        String(RGB(red_amount, 255, 255)) + &
        ")'"
```

The following is a simpler example without the If function. You do not need quotes around the value if you are not specifying an expression. Here the String and RGB functions produce a constant value in the resulting modstring:

```
modstring = "emp_id.Color=" + &
        String(RGB(red_amount, 255, 255))
```

You can set several properties with a single call to Modify by including each property setting in *modstring* separated by spaces. For example, assume the following is entered on a single line in the script editor:

```
rtn = dw_1.Modify("emp_id.Font.Italic=0
oval_1.Background.Mode=0
oval_1.Background.Color=255")
```

However, it is easier to understand and debug a script in which each call to Modify sets one property.

Debugging tip If you build your *modstring* and store it in a variable that is the argument for Modify, you can look at the value of the variable in Debug mode. When Modify's error message reports a column number, you can count the characters as you look at the compiled *modstring*.

Modifying a WHERE clause For efficiency, use Modify instead of SetSQLSelect to modify a WHERE clause. Modify is faster because it does not verify the syntax and does not change the update status of the DataWindow object. However, Modify is more susceptible to user error. SetSQLSelect modifies the syntax twice (when the syntax is modified and when the retrieve executes) and affects the update status of the DataWindow object.

PocketBuilder already includes many functions for modifying a DataWindow. Before using Modify, check the list of DataWindow functions in *Objects and Controls* to see if a function exists for making the change. Many of these functions are listed in the See also section.

Modify is for modifying the properties of a DataWindow *object* and its internal controls. You can set properties of the DataWindow *control* that contains the object using standard dot notation. For example, to put a border on the control, specify:

```
dw_1.Border = TRUE
```

Examples

These examples illustrate the typical uses listed in the Usage section. The examples use PowerScript. For a discussion of Modify and nested quotation marks in JavaScript, see Chapter 5, "Accessing DataWindow Object Properties in Code."

Changing colors The effect of setting the Color property depends on the control you are modifying. To set the background color of the whole DataWindow object, use the following syntax:

dwcontrolname.**Modify** ("DataWindow.Color='*long*'")

To set the text color of a column or a text control, use similar syntax:

dwcontrolname.**Modify** ("*controlname*.Color='*long*'")

To set the background color of a column or other control, use the following syntax to set the mode and color. Make sure the mode is opaque:

dwcontrolname.**Modify** ("*controlname*.Background.Mode= &
'<0 - Opaque, 1 - Transparent>'")

dwcontrolname.**Modify** ("*controlname*.Background.Color='*long*'")

The following examples use the syntaxes shown above to set the colors of various parts of the DataWindow object.

This statement changes the background color of the DataWindow dw_cust to red:

```
dw_cust.Modify("DataWindow.Color = 255")
```

This statement causes the DataWindow dw_cust to display the text of values in the salary column in red if they exceed 90,000 and in green if they do not:

```
dw_cust.Modify( &
       "salary.Color='0~tIf(salary>90000,255,65280)'")
```

This statement nests one If function within another to provide three possible colors. The setting causes the DataWindow dw_cust to display the department ID in green if the ID is 200, in red if it is 100, and in black if it is neither:

```
dw_cust.Modify("dept_id.Color='0~t " &
     + "If(dept_id=200,65380,If(dept_id=100,255,0))'")
```

The following example uses a complex expression with nested If functions to set the background color of the salary column according to the salary values. Each portion of the concatenated string is shown on a separate line. See the pseudocode in the comments for an explanation of what the nested If functions do. The example also sets the background mode to opaque so that the color settings are visible. The example includes error checking, which displays Modify's error message, if any:

```
string mod_string, err
long color1, color2, color3, default_color

err = dw_emp.Modify("salary.Background.Mode=0")
IF err <> "" THEN
       MessageBox("Status", &
          "Change to Background Mode Failed " + err)
       RETURN
END IF

/* Pseudocode for mod_string:
If salary less than 10000, set the background to red.
If salary greater than or equal to 10000 but less than
20000, set the background to blue.
If salary greater than or equal to 20000 but less than
30000, set the background color to green.
Otherwise, set the background color to white, which is
also the default.
*/
color1 = 255 //red
color2 = 16711680 //blue
color3 = 65280 //green
default_color = 16777215//white

mod_string = &
       "salary.Background.Color = '" &
          + String(default_color) &
          + "~tIf(salary < 10000," &
          + String(color1) &
          + ",If(salary < 20000," &
          + String(color2) &
          + ",If(salary < 30000," &
          + String(color3) &
          + "," &
          + String(default_color) &
          + ")))'"

err = dw_emp.Modify(mod_string)
IF err <> "" THEN
       MessageBox("Status", &
          "Change to Background Color Failed " + err)
       RETURN
END IF
```

This example sets the text color of a RadioButton column to the value of color1 (red) if the column's value is Y; otherwise, the text is set to black. As above, each portion of the concatenated string is shown on a separate line:

```
integer color1, default_color
string mod_string, err

color1 = 255 //red
default_color = 0 //black

mod_string = "yes_or_no.Color ='" &
       + String(default_color) &
       + "~tif(yes_or_no=~~'Y~~'," &
       + String(color1) &
       + "," &
       + String(default_color) &
       + ")'"
err = dw_emp.Modify(mod_string)
IF err <> "" THEN
       MessageBox("Status", &
           "Modify to Text Color " &
           + "of yes_or_no Failed " + err)
       RETURN
END IF
```

Changing displayed text To set the text of a text control, the next two examples use this syntax:

dwcontrolname.**Modify** ("*textcontrolname*.Text='string"')

This statement changes the text in the text control Dept_t in the DataWindow dw_cust to Dept:

```
dw_cust.Modify("Dept_t.Text='Dept'")
```

This statement sets the displayed text of dept_t in the DataWindow dw_cust to Marketing if the department ID is greater than 201; otherwise it sets the text to Finance:

```
dw_cust.Modify("dept_t.Text='none~t " + &
       "If(dept_id > 201,~'Marketing~',~'Finance~')'")
```

Updating more than one table An important use of Modify is to make it possible to update more than one table from one DataWindow object. The following script updates the table that was specified as updatable in the DataWindow painter; then it uses Modify to make the other joined table updatable and to specify the key column and which columns to update. This technique eliminates the need to create multiple DataWindow objects or to use embedded SQL statements to update more than one table.

In this example, the DataWindow object joins two tables: department and employee. First department is updated, with status flags not reset. Then employee is made updatable and is updated. If all succeeds, the Update command resets the flags and COMMIT commits the changes. Note that to make the script repeatable in the user's session, you must add code to make department the updatable table again:

```
integer rc
string err

/* The SELECT statement for the DataWindow is:
SELECT department.dept_id, department.dept_name,
employee.emp_id, employee.emp_fname,
employee.emp_lname FROM department, employee ;
*/

// Update department, as set up in the DW painter
rc = dw_1.Update(TRUE, FALSE)

IF rc = 1 THEN
       //Turn off update for department columns.
       dw_1.Modify("department_dept_name.Update = No")
       dw_1.Modify("department_dept_id.Update = No")
       dw_1.Modify("department_dept_id.Key = No")

       // Make employee table updatable.
       dw_1.Modify( &
        "DataWindow.Table.UpdateTable = ~"employee~"")

       //Turn on update for desired employee columns.
       dw_1.Modify("employee_emp_id.Update = Yes")
       dw_1.Modify("employee_emp_fname.Update = Yes")
       dw_1.Modify("employee_emp_lname.Update = Yes")
       dw_1.Modify("employee_emp_id.Key = Yes")

       //Then update the employee table.
       rc = dw_1.Update()
       IF rc = 1 THEN
          COMMIT USING SQLCA;
       ELSE
       ROLLBACK USING SQLCA;
       MessageBox("Status", &
          + "Update of employee table failed. " &
          + "Rolling back all changes.")
       END IF
ELSE
```

```
        ROLLBACK USING SQLCA;
        MessageBox("Status", &
            + "Update of department table failed. " &
            + "Rolling back changes to department.")
END IF
```

Adding a WHERE clause The following scripts dynamically add a WHERE clause to a DataWindow object that was created with a SELECT statement that did not include a WHERE clause. (Since this example appends a WHERE clause to the original SELECT statement, additional code would be needed to remove a where clause from the original SELECT statement if it had one.) This technique is useful when the arguments in the WHERE clause might change at execution time.

The original SELECT statement might be:

```
SELECT employee.emp_id, employee.l_name
       FROM employee
```

Presumably, the application builds a WHERE clause based on the user's choices. The WHERE clause might be:

```
WHERE emp_id > 40000
```

The script for the window's Open event stores the original SELECT statement in original_select, an instance variable:

```
dw_emp.SetTransObject(SQLCA)
original_select = &
       dw_emp.Describe("DataWindow.Table.Select")
```

The script for a CommandButton's Clicked event attaches a WHERE clause stored in the instance variable where_clause to original_select and assigns it to the DataWindow's Table.Select property:

```
string rc, mod_string

mod_string = "DataWindow.Table.Select='" &
       + original_select + where_clause + "'"

rc = dw_emp.Modify(mod_string)
IF rc = "" THEN
       dw_emp.Retrieve( )
ELSE
       MessageBox("Status", "Modify Failed" + rc)
END IF
```

Quotes inserted in the DataWindow painter
For SQL Anywhere, the DataWindow painter puts double quotes around the table and column name (for example, SELECT "EMPLOYEE"."EMP_LNAME"). Unless you have removed the quotes, the sample WHERE clause must also use these quotes. For example:

```
where_clause = &
       " where ~~~"EMPLOYEE~~~".~~~"SALARY~~~" > 40000"
```

Query mode Query mode provides an alternate view of a DataWindow in which the user specifies conditions for selecting data. PocketBuilder builds the WHERE clause based on the specifications. When the user exits query mode, you can retrieve data based on the modified SELECT statement.

In this example, a window that displays a DataWindow control has a menu that includes a selection called Select Data. When the user chooses it, its script displays the DataWindow control in query mode and checks the menu item. When the user chooses it again, the script turns query mode off and retrieves data based on the new WHERE clause specified by the user through query mode. The script also makes a CheckBox labeled Sort data visible, which turns query sort mode on and off.

The script for the Select Data menu item is:

```
string rtn

IF m_selectdata.Checked = FALSE THEN
       // Turn on query mode so user can specify data
       rtn = dw_1.Modify("DataWindow.QueryMode=YES")

       IF rtn = "" THEN
          // If Modify succeeds, check menu to show
          // Query mode is on and display sort CheckBox
          This.Check()
          ParentWindow.cbx_sort.Show()
       ELSE
          MessageBox("Error", &
              "Can't access query mode to select data.")
       END IF
ELSE
       // Turn off Query mode and retrieve data
       // based on user's choices
       rtn = dw_1.Modify("DataWindow.QueryMode=NO")
```

```
        IF rtn = "" THEN
            // If Modify succeeds, uncheck menu to show
            // Query mode is off, hide the sort
            // CheckBox, and retrieve data
            This.UnCheck()
            ParentWindow.cbx_sort.Hide()
            dw_1.AcceptText()
            dw_1.Retrieve()
        ELSE
            MessageBox("Error", &
                "Failure exiting query mode.")
        END IF
END IF
```

A simple version of the script for Clicked event of the Sort data CheckBox follows. You could add code as shown in the Menu script above to check whether Modify succeeded:

```
IF This.Checked = TRUE THEN
       dw_1.Modify("DataWindow.QuerySort=YES")
ELSE
       dw_1.Modify("DataWindow.QuerySort=NO")
END IF
```

For details on how you or the user specifies information in query mode, see the *Users Guide*.

DataWindow presentation styles
You cannot use QueryMode and QuerySort with DataWindow objects that use any of the following presentation styles: N-Up, Label, Crosstab, RichText, and Graph.

Prompt for criteria is another way of letting the user specify retrieval criteria. You set it on a column-by-column basis. When a script retrieves data, PocketBuilder displays the Specify Retrieval Criteria window, which gives the user a chance to specify criteria for all columns that have been set.

In a script that is run before you retrieve data, for example, in the Open event of the window that displays the DataWindow control, the following settings would make the columns emp_name, emp_salary, and dept_id available in the Specify Retrieval Criteria dialog when the Retrieve method is called:

```
dw_1.Modify("emp_name.Criteria.Dialog=YES")
dw_1.Modify("emp_salary.Criteria.Dialog=YES")
dw_1.Modify("dept_id.Criteria.Dialog=YES")
```

There are other Criteria properties that affect both query mode and prompt for criteria. For details, see the Criteria DataWindow object property in Chapter 3, "DataWindow Object Properties."

Retrieve as needed In this example, the DataWindow object has been set up with Retrieve Only As Needed selected. When this is on, PocketBuilder retrieves enough rows to fill the DataWindow, displays them quickly, then waits for the user to try to display additional rows before retrieving more rows. If you want the fast initial display but do not want to leave the cursor open on the server, you can turn off Retrieve Only As Needed with Modify.

After you have determined that enough rows have been retrieved, the following code in the RetrieveRow event script changes the Retrieve.AsNeeded property, which forces the rest of the rows to be retrieved:

```
dw_1.Modify("DataWindow.Retrieve.AsNeeded=NO")
```

Changing the data source This example changes the data source of a DataWindow object from a SQL SELECT statement to a stored procedure. This technique works *only* if the result set does not change (that is, the number, type, and order of columns is the same for both sources).

When you define the DataWindow object, you must define all possible DataWindow retrieval arguments. In this example, the SELECT statement defined in the painter has three arguments, one of type string, one of type number, and one of type date. The stored procedure has two arguments, both of type string. So, in the painter, you need to define four DataWindow arguments, two of type string, one of type number, and one of type date. (Note that you do not have to use all the arguments you define.)

```
string rc, mod_string, name_str = "Watson"
integer dept_num = 100

// Remove the DataWindow's SELECT statement
Dw_1.Modify("DataWindow.Table.Select = ''")

// Set the Procedure property to your procedure
mod_string = "DataWindow.Table.Procedure = &
       '1 execute dbo.emp_arg2;1 @dept_id_arg &
       = :num_arg1, @lname_arg = :str_arg1'"
rc = dw_1.Modify(mod_string)

// If change is accepted, retrieve data
IF rc = "" THEN
       dw_1.Retrieve(dept_num, name_str)
```

```
ELSE
       MessageBox("Status", &
          "Change to DW Source Failed " + rc)
END IF
```

Replacing a DropDownDataWindow object
Suppose you use Modify to replace one DropDownDataWindow object with another; for example:

```
dw_parent.Modify(dept_id.dddw.name= &
       d_dddw_empsal_by_dept )
```

PocketBuilder compares the two DataWindow objects and reuses the original result set if the number of columns and their datatypes match. The display and data value column names must exist in the data object SQL statements for both objects. If there are any differences, PocketBuilder will re-retrieve the data.

Deleting and adding controls in the DataWindow object This statement deletes a bitmap control called logo from the DataWindow dw_cust:

```
dw_cust.Modify("destroy logo")
```

This statement deletes the column named salary from the DataWindow dw_cust. Note that this example includes the keyword column, so the column in the DataWindow and the data are both deleted:

```
dw_cust.Modify("destroy column salary")
```

This example adds a rectangle named rect1 to the header area of the DataWindow dw_cust (with the value of modstring entered as a single line):

```
string modstring

modstring = 'create rectangle(Band=background X="206"
Y="6" height="69" width="1363" brush.hatch="6"
brush.color="12632256" pen.style="0" pen.width="14"
pen.color="268435584" background.mode="2"
background.color="-1879048064" name=rect1 )'

dw_cust.Modify(modstring)
```

These statements add a bitmap named logo to the header area for grouping level 1 in the DataWindow dw_cust (with the value of modstring entered as a single line):

```
string modstring

modstring = 'create bitmap(band=footer x="37" y="12"
height="101" width="1509" filename="C:\PB\BEACH.BMP"
border="0" name=bmp1 )'

dw_cust.Modify(modstring)
```

Syntax for creating controls

To create a control, you must provide DataWindow syntax. The easiest way to get correct syntax for all the necessary properties is to paint the control in the DataWindow painter and export the syntax to a file. Then you make any desired changes and put the syntax in your script, as shown above. This is the only way to get accurate syntax for complex controls like graphs.

See also

Describe
Reset
SetBorderStyle
SetDataStyle
SetFilter
SetFormat
SetPosition
SetRowFocusIndicator
SetSeriesStyle
SetSQLPreview
SetSQLSelect
SetTabOrder
SetValidate

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	✓

Syntax

integer*objectname*.**Move** (integer *x*, integer *y*)

Argument	Description
objectname	A reference to an object or control you want to move
x	The x coordinate of the new location in PowerBuilder units
y	The y coordinate of the new location in PowerBuilder units

Return value

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

Inherited from system window object. For information, see Move in the *PowerScript Reference*.

OLEActivate

Description

Activates Object Linking and Embedding (OLE) for the specified object and sends the specified command verb to the OLE server application.

PocketBuilder	✗
PowerBuilder	✓

Syntax

PowerBuilder DataWindow control or child DataWindow

integer *dwcontrol*.**OLEActivate** (long *row*, integer *column*, integer *verb*)

integer *dwcontrol*.**OLEActivate** (long *row*, string *column*, integer *verb*)

Return value

Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is NULL, OLEActivate returns NULL.

Paste

Description

Inserts (pastes) the contents of the clipboard into the specified control. If no text is selected in the control, the text on the clipboard is pasted at the insertion point. If text is selected, Paste replaces the selected text with the text on the clipboard.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

long *dwcontrol*.**Paste** ( )

Argument	Description
dwcontrol	A reference to a DataWindow control. Text is pasted into the edit control over the current row and column.

Return value

Returns the number of characters that were pasted into the edit control for *dwcontrol*. If nothing has been cut or copied (the clipboard is empty), Paste 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.

If *dwcontrol* is NULL, the method returns NULL.

Usage

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 *dwcontrol* or to replace selected text with a specific string, use the ReplaceText method.

Using with other controls
For use of this method with other PocketBuilder controls, see Paste in the *PowerScript Reference*.

Examples

If the clipboard contains "Proposal good for 90 days" and no text is selected in the edit control of dw_rpt, this statement pastes "Proposal good for 90 days" at the insertion point in the edit control and returns 25:

```
dw_rpt.Paste()
```

If the clipboard contains the string “Final Edition”, the edit control in dw_rpt contains “This is a Preliminary Draft”, and the text in edit control is selected, this statement deletes “This is a Preliminary Draft”, replaces it with “Final Edition”, and returns 13:

```
dw_rpt.Paste()
```

See also

Copy
Cut
ReplaceText

PasteRTF

Description

Pastes rich text data from a string into a DataWindow control or DataStore object.

PocketBuilder	✗
PowerBuilder	✓

Syntax

PowerBuilder DataWindow control or DataStore object

long *rtename*.**PasteRTF** (string *richtextstring*, { Band *band* }

Return value

Returns the number of characters pasted if it succeeds and -1 if an error occurs. If *richtextstring* is NULL, PasteRTF returns NULL.

PointerX

Description

Determines the distance of the pointer from the left edge of the specified object.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

integer *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

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.

Usage

Inherited from DragObject. For information, see PointerX in the *PowerScript Reference*.

PointerY

Description

Determines the distance of the pointer from the top of the specified object.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

integer *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

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.

Usage

Inherited from DragObject. For information, see PointerY in the *PowerScript Reference*.

Position

Reports the position of the insertion point in a DataWindow.

To report	Use
The position of the insertion point in a DataWindow that does not have a RichTextEdit presentation style	Syntax 1
The position of the insertion point or the start and end of selected text in a DataWindow whose object has the RichTextEdit presentation style	Syntax 2

Syntax 1 For DataWindows with standard presentation styles

Description

Determines the position of the insertion point in an edit control.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

long *editname*.**Position** ( )

Argument	Description
editname	A reference to a DataWindow control in which you want to find the location of the insertion point

Return value

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.

Position reports the insertion point's position in the edit control over the current row and column.

Using with other controls
For use of this method with other PocketBuilder controls, see Position in the *PowerScript Reference*.

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 DataWindows with RichTextEdit presentation styles

Description

Determines the line and column position of the insertion point or the start and end of selected text in a RichTextEdit control.

PocketBuilder	✗
PowerBuilder	✓

Syntax

PowerBuilder

band *rtename*.**Position** (long *fromline*, long *fromchar* {, long *toline*, long *tochar* }

Return value

Returns the band containing the selection or insertion point. The returned value is a value of the Band enumerated datatype (Detail!, Header!, or Footer!).

PostEvent

Description

Adds an event to the end of the event queue of an object.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

boolean *objectname*.**PostEvent** (TrigEvent *event*, { long *word*, long *long* })

boolean *objectname*.**PostEvent** (TrigEvent *event*, { long *word*, string *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*.
word (optional)	A 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 for *word*, enter 0. (For cross-platform compatibility, WordParm and LongParm are both longs.)
long (optional)	A value 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

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

Inherited from PowerObject. For information, see PostEvent in the *PowerScript Reference*.

Print

Sends data to the current printer (or spooler, if the user has a spooler set up). There are two syntaxes that you can use with DataWindows:

To	Use
Send the contents of a DataWindow control or DataStore to the printer as a print job.	Syntax 1
Include a visual object, such as a window or a graph control in a print job. For a description of system print commands, see the *PowerScript Reference*.	Syntax 2

Third-party software required for printing
You must install the FieldSoftware PrinterCE SDK before you can use print methods in PocketBuilder applications deployed to a Pocket PC device. An evaluation version of this software is available from the FieldSoftware Web site at http://www.fieldsoftware.com.

Syntax 1 For printing a single DataWindow or DataStore

Description

Sends the contents of a DataWindow control or DataStore object to the printer as a print job.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

integer *dwcontrol*.**Print** ({ boolean *canceldialog* })

Argument	Description
dwcontrol	The name of the DataWindow control, DataStore, or child DataWindow that contains the information to be printed.
canceldialog (optional)	A boolean value indicating whether you want to display a nonmodal dialog that allows the user to cancel printing. Values are: • TRUE — (Default) Display the dialog. • FALSE — Do not display the dialog. **Working with DataStore objects** When working with DataStores, the *canceldialog* argument must always be set to FALSE.

Return value

Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is NULL, Print returns NULL.

Usage

Printed output uses the same fonts and layout that appear on screen for the DataWindow object.

Printing multiple DataWindows in a single job

PocketBuilder manages a print job by opening the job, sending data, and closing the job. When you use Syntax 1, print job management happens automatically. You do not need to use the PrintOpen and PrintClose functions.

Use Syntax 1 to print the contents of a single DataWindow object. The Print method prints all the rows that have been retrieved. To print multiple DataWindows as a single job, do not use Print. Instead, open the print job with PrintOpen, call the PowerScript system function PrintDataWindow for each DataWindow, and close the job.

Events for DataWindow printing

When you use Print for DataWindow controls or DataStores, it triggers a PrintStart event just before any data is sent to the printer (or spooler), a PrintPage event for each page break, and a PrintEnd event when printing is complete.

The PrintPage event has return codes that let you control whether to print the page about to be formatted. You can skip the upcoming page by returning a value of 1 in the PrintPage event.

Examples

This statement sends the contents of dw_employee to the current printer:

```
dw_employee.Print()
```

See also

PrintDataWindow in the *PowerScript Reference*

Syntax 2 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	✓

Syntax

integer *objectname*.**Print** (long *printjobnumber*, integer *x*, integer *y* {, integer *width*, integer *height* })

Argument	Description
objectname	The name of the object that you want to print. The object must be either 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

Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is NULL, Print returns NULL.

Usage

PocketBuilder manages a print job by opening the job, sending data, and closing the job. When you use Syntax 2, you must call the PrintOpen function and the PrintClose or PrintCancel functions yourself to manage the process. For more information, see the *PowerScript Reference*.

Print area and margins

The print area is the physical page size minus any margins in the printer itself. Depending on the printer, you might be able to change margins using PrintSend and printer-defined escape sequences.

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

Print in the *PowerScript Reference*
PrintCancel
PrintClose in the *PowerScript Reference*
PrintOpen in the *PowerScript Reference*
PrintScreen in the *PowerScript Reference*

PrintCancel

Cancels printing and deletes the spool file, if any. There are two syntaxes.

To	Use
Cancel printing of a DataWindow or DataStore printed with the Print function.	Syntax 1
Cancel a print job that you began with the PrintOpen function. For a description of PocketBuilder system print commands, see the *PowerScript Reference*.	Syntax 2

Syntax 1 For DataWindows and DataStores

Description

Cancels the printing of a DataWindow or DataStore that was printed using Syntax 1 of Print.

PocketBuilder	✗
PowerBuilder	✓

Syntax

PowerBuilder DataWindow control, DataStore object, or child DataWindow

integer *dwcontrol*.**PrintCancel** ( )

Return value

Returns 1 if it succeeds and -1 if an error occurs. If *dwcontrol* is NULL, PrintCancel returns NULL.

Syntax 2 For canceling a print job

Description

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

PowerBuilder DataWindow control

integer **PrintCancel** (long *printjobnumber*)

Return value

Returns 1 if it succeeds and -1 if an error occurs. If *printjobnumber* is NULL, PrintCancel returns NULL.

ReplaceText

Description

Replaces selected text in the edit control for the current row and column with a specified string.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

long *editname*.**ReplaceText** (string *string*)

Argument	Description
editname	A reference to a DataWindow control
string	The string that replaces the selected text

Return value

Returns the number of characters in *string* and -1 if an error occurs. If any argument's value is NULL, the method returns NULL.

Usage

If there is no selection, ReplaceText inserts the replacement text at the cursor position. To use the contents of the clipboard as the replacement text, call the Paste method instead of ReplaceText.

Using with other controls
For use of this method with other PocketBuilder controls, see ReplaceText in the *PowerScript Reference*.

Examples

If the DataWindow edit control 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 text in the edit control is “Offer Good for 60 Days”:

```
dw_salesoffer.ReplaceText("60 Days")
```

If there is no selected text, this statement inserts “New product” at the cursor position in the edit control for dw_products:

```
dw_products.ReplaceText("New product")
```

See also

Copy
Cut
Paste
ReplaceText in the *PowerScript Reference*

ReselectRow

Description

Accesses the database to retrieve values for all columns that can be updated and refreshes all timestamp columns in a row in a DataWindow control or DataStore. The values from the database are redisplayed in the row.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

integer *dwcontrol*.**ReselectRow** (long *row*)

Argument	Description
dwcontrol	A reference to the DataWindow control, DataStore, or child DataWindow in which you want to reselect a row
row	A value identifying the row to reselect

Return value

Returns 1 if it is successful and -1 if the row cannot be reselected (for example, the DataWindow object cannot be updated or the row was deleted by another user). If any argument’s value is NULL, the method returns NULL.

Usage

ReselectRow is supported for SQLSelect DataWindows. Use ReselectRow to discard values the user changed and replace them with values from the database after an update fails (due to a concurrent access error, for example).

About timestamp support
Timestamp support is not available in all DBMSs. For information on timestamp columns, see the documentation for your DBMS.

Examples

This statement reselects row 5 in the DataWindow control dw_emp:

```
dw_emp.ReselectRow(5)
```

This statement reselects the clicked row if the update is not successful:

```
IF dw_emp.Update( ) < 0 THEN
       dw_emp.ReselectRow(dw_emp.GetClickedRow())
END IF
```

See also

GetClickedRow
SelectRow
Update

Reset

Description

Clears all the data from a DataWindow control or DataStore object.

For the syntax to use for deleting graphs within a DataWindow object that have an external data source, see Reset on page 684. For the syntax to use with other PocketBuilder controls, see Reset in the *PowerScript Reference*.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

integer *dwcontrol*.**Reset** ( )

Argument	Description
dwcontrol	A reference to a DataWindow control, DataStore, or child DataWindow

Return value

Returns 1 if it succeeds and -1 if an error occurs. The return value is usually not used. If *dwcontrol* is NULL, the method returns NULL.

Usage

Reset is not the same as deleting rows from the DataWindow object or child DataWindow. Reset affects the application only, not the database. If you delete rows and then call the Update method, the rows are deleted from the database table associated with the DataWindow object. If you call Reset and then call Update, no changes are made to the table.

Preventing rows from being retrieved after calling Reset

If you call Reset when the Retrieve As Needed option is set, Reset will clear the rows that have been retrieved. However, because Retrieve As Needed is on, the DataWindow immediately retrieves the next set of rows.

To prevent the rows from being retrieved, call DBCancel before calling Reset. If all the rows have been retrieved (the cursor has been closed and the RetrieveEnd event has occurred), then when Reset clears the DataWindow, it stays empty.

Examples

This statement completely clears the contents of dw_employee:

```
dw_employee.Reset()
```

In a DataWindow whose Retrieve As Needed option is on, this example cancels the retrieval before resetting the DataWindow:

```
dw_employee.DBCancel()
dw_employee.Reset()
```

See also

DeleteRow

ResetTransObject

Description

Stops a DataWindow control or DataStore from using the programmer-specified transaction object that is currently in effect through a call to the SetTransObject method. After you call the ResetTransObject method, the DataWindow control or DataStore uses its internal transaction object.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

integer *dwcontrol*.**ResetTransObject** ( )

Argument	Description
dwcontrol	A reference to a DataWindow control, DataStore, or child DataWindow

Return value

Returns 1 if it succeeds and -1 if an error occurs. The return value is usually not used.

If *dwcontrol* is NULL, the method returns NULL.

Usage

If you reset the transaction object and SetTrans has never been called to set the values in the internal transaction object, call SetTrans to set them or SetTransObject to establish a new programmer-specified transaction object.

ResetTransObject is almost never used because programmer-specified and internal transaction objects in one application are generally not used together. Programmer-specified transaction objects, specified with SetTransObject, provide better application performance. To change the programmer-specified transaction object, simply call SetTransObject again.

Examples

This statement stops dw_employee from using programmer-specified transaction objects:

```
dw_employee.ResetTransObject()
```

See also

GetTrans
SetTrans
SetTransObject

ResetUpdate

Description

Clears the update flags in the primary and filter buffers and empties the delete buffer of a DataWindow or DataStore.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

integer *dwcontrol*.**ResetUpdate** ( )

Argument	Description
dwcontrol	The name of the DataWindow control, DataStore, or child DataWindow in which you want to reset the update flags

Return value

Returns 1 if it succeeds and -1 if an error occurs. If *dwcontrol* is NULL, the method returns NULL.

Usage

When a row is changed, inserted, or deleted, its update flag is set, making it marked for update. By default the Update method turns these flags off. If you want to coordinate updates of more than one DataWindow or DataStore, however, you can prevent Update from clearing the flags. Then, after you verify that all the updates succeeded, you can call ResetUpdate for each DataWindow to clear the flags. If one of the updates failed, you can keep the update flags, prompt the user to fix the problem, and try the updates again.

You can find out which rows are marked for update with the GetItemStatus method. If a row is in the delete buffer or if it is in the primary or filter buffer and has NewModified! or DataModified! status, its update flag is set. After update flags are cleared, all rows have the status NotModified! or New! and the delete buffer is empty.

Examples

These statements coordinate the update of two DataWindow objects:

```
int rtncode
CONNECT USING SQLCA;
dw_cust.SetTransObject(SQLCA)
dw_sales.SetTransObject(SQLCA)

rtncode = dw_cust.Update(TRUE, FALSE)
IF rtncode = 1 THEN
       rtncode = dw_sales.Update(TRUE, FALSE)
       IF rtncode = 1 THEN
           dw_cust.ResetUpdate() // Both updates are OK
           dw_sales.ResetUpdate()// Clear update flags
           COMMIT USING SQLCA;   // Commit them
       ELSE
           ROLLBACK USING SQLCA; // 2nd update failed
       END IF
END IF
```

See also

Update

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	✓

Syntax

integer *objectname*.**Resize** (integer *width*, integer *height*)

Argument	Description
objectname	A reference to the object or control you want to resize
width	The new width in PowerBuilder units
height	The new height in PowerBuilder units

Return value

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, the method returns NULL.

Usage

You cannot use Resize for a child DataWindow.

Use with other PocketBuilder objects and controls

Resize does not resize a minimized or maximized window. If the window is minimized or maximized, Resize returns –1.

For use with other PocketBuilder controls, see Resize in the *PowerScript Reference*.

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)
```

Retrieve

Description

Retrieves rows from the database for a DataWindow control or DataStore. If arguments are included, the argument values are used for the retrieval arguments in the SQL SELECT statement for the DataWindow object or child DataWindow.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

long *dwcontrol*.**Retrieve** ({ any *argument*, any *argument* . . . })

Argument	Description
dwcontrol	A reference to a DataWindow control, DataStore, or child DataWindow
argument (optional)	One or more values that you want to use as retrieval arguments in the SQL SELECT statement defined in *dwcontrol*

Return value

Returns the number of rows displayed (that is, rows in the primary buffer) if it succeeds and -1 if it fails. If there is no DataWindow object assigned to the DataWindow control or DataStore, this method returns -1.

Usage

After rows are retrieved, the DataWindow object's filter is applied. Therefore, any retrieved rows that do not meet the filter criteria are immediately moved to the filter buffer and are not included in the return count.

Before you can retrieve rows for a DataWindow control or DataStore, you must specify a transaction object with SetTransObject or SetTrans. If you use SetTransObject, you must also use a SQL CONNECT statement to establish a database connection.

Normally, when you call Retrieve, any rows that are already in the DataWindow control or DataStore are discarded and replaced with the retrieved rows. You can return the code 2 in the RetrieveStart event to prevent this. In this case, Retrieve adds any retrieved rows to the ones that already exist in the buffers.

Retrieval arguments If arguments are expected but not specified, the user is prompted for the retrieval arguments.

A retrieval argument can be NULL if the SELECT statement is designed to handle null values. For example, if a two-part WHERE clause is separated by OR, then either part can be NULL while the other matches values in the database.

Events Retrieve may trigger these events:

DBError
RetrieveEnd
RetrieveRow
RetrieveStart

Examples

This statement causes dw_emp1 to retrieve rows from the database.

```
dw_emp1.Retrieve()
```

This example illustrates how to set up a connection and then retrieve rows in the DataWindow control. A typical scenario is to establish the connection in the application's Open event and to retrieve rows in the Open event for the window that contains the DataWindow control.

The following is a script for the application open event. SQLCA is the default transaction object. The ProfileString function is getting information about the database connection from an initialization file:

```
// Set up Transaction object from the INI file
SQLCA.DBMS = ProfileString("myapp.ini", &
       "Database", "DBMS", " ")
SQLCA.DbParm = ProfileString("myapp.ini", &
       "Database", "DbParm", " ")
// Connect to database
CONNECT USING SQLCA;
// Test whether the connect succeeded
IF SQLCA.SQLCode <> 0 THEN
       MessageBox("Connect Failed", &
           "Cannot connect to database." &
           + SQLCA.SQLErrText)
       RETURN
END IF
Open(w_main)
```

To continue the example, the open event for w_main sets the transaction object for the DataWindow control dw_main to SQLCA and retrieves rows from the database. If no rows were retrieved or if there is an error (that is, the return value is negative), the script displays a message to the user:

```
long ll_rows
dw_main.SetTransObject(SQLCA)
ll_rows = dw_main.Retrieve()
IF ll_rows < 1 THEN MessageBox( &
       "Database Error", &
       "No rows retrieved.")
```

This example illustrates the use of retrieval arguments. Assume that :Salary and :Region are declared as arguments in the DataWindow painter and dw_emp has this SQL SELECT statement:

```
SELECT Name, emp.sal, sales.rgn From Employee, Sales
       WHERE emp.sal > :Salary and sales.rgn = :Region
```

Then this statement causes dw_emp1 to retrieve employees from the database who have a salary greater than $50,000 and are in the northwest region:

```
dw_1.Retrieve(50000, "NW")
```

This example also illustrates retrieval arguments. Assume dw_EmpHist contains this SQL SELECT statement and emps is defined as a number array:

```
SELECT EmpNbr, Sal, Rgn From Employee
       WHERE EmpNbr IN (:emps)
```

These statements cause dw_EmpHist to retrieve Employees from the database whose employee numbers are values in the array emps:

```
Double emps[3]
emps[1] = 100
emps[2] = 200
emps[3] = 300
dw_EmpHist.Retrieve(emps)
```

The following example illustrates how to use Retrieve twice to get data meeting different criteria. Assume the SELECT statement for the DataWindow object requires one argument, the department number. Then these statements retrieve all rows in the database in which department number is 100 or 200.

The script for the RetrieveStart event in the DataWindow control sets the return code to 2 so that the rows and buffers of the DataWindow control will not be cleared before each retrieval:

```
RETURN 2
```

The script for the Clicked event for a Retrieve CommandButton retrieves the data with two function calls. The Reset method clears any previously retrieved rows, normally done by Retrieve. Here, Retrieve is prevented from doing it by the return code in the RetrieveStart event:

```
dw_1.Reset( )
dw_1.Retrieve(100)
dw_1.Retrieve(200)
```

See also

DeleteRow
InsertRow
SetTrans
SetTransObject
Update

RowCount

Description

Obtains the number of rows that are currently available in a DataWindow control or DataStore. To determine the number of rows available, the RowCount method checks the primary buffer.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

long *dwcontrol*.**RowCount** ( )

Argument	Description
dwcontrol	A reference to a DataWindow control, DataStore, or child DataWindow

Return value

Returns the number of rows that are currently available in *dwcontrol*, 0 if no rows are currently available, and -1 if an error occurs. If *dwcontrol* is NULL, the method returns NULL.

Usage

The primary buffer for a DataWindow control or DataStore contains the rows that are currently available for display or printing. These are the rows counted by RowCount. The number of currently available rows equals the total number of rows retrieved minus any deleted or filtered rows plus any inserted rows. The deleted and filtered rows are stored in the DataWindow's delete and filter buffers.

Examples

This statement returns the number of rows currently available in dw_Employee:

```
long NbrRows
NbrRows = dw_Employee.RowCount()
```

This example determines when the user has scrolled to the end of a DataWindow control. It compares the row count with the DataWindow property LastRowOnPage:

```
dw_1.ScrollNextPage()
IF dw_1.RowCount() = Integer(dw_1.Describe( &
          "DataWindow.LastRowOnPage")) THEN
       ... // Appropriate processing
END IF
```

See also

DeleteRow
DeletedCount
Filter
FilteredCount
InsertRow
ModifiedCount
SetFilter
Update

RowsCopy

Description

Copies a range of rows from one DataWindow control (or DataStore object) to another, or from one buffer to another within a single DataWindow control (or DataStore).

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

integer *dwcontrol*.**RowsCopy** (long *startrow*, long *endrow*, DWBuffer *copybuffer*, datawindow *targetdw*, long *beforerow*, DWBuffer *targetbuffer*)

integer *dwcontrol*.**RowsCopy** (long *startrow*, long *endrow*, DWBuffer *copybuffer*, datastore *targetdw*, long *beforerow*, DWBuffer *targetbuffer*)

integer *dwcontrol*.**RowsCopy** (long *startrow*, long *endrow*, DWBuffer *copybuffer*, datawindowchild *targetdw*, long *beforerow*, DWBuffer *targetbuffer*)

Argument	Description
dwcontrol	The name of the DataWindow control, DataStore, or child DataWindow from which you want to copy rows.
startrow	The number of the first row you want to copy.

Argument	Description
endrow	The number of the last row you want to copy.
copybuffer	A value of the dwBuffer enumerated datatype identifying the DataWindow buffer from which you want to copy rows. For a list of valid values, see DWBuffer on page 372.
targetdw	A reference to the DataWindow control or DataStore object to which you want to copy the rows. *Targetdw* can be the same DataWindow (or DataStore) or another DataWindow (or DataStore).
beforerow	The number of the row before which you want to insert the copied rows. To insert after the last row, use any value that is greater than the number of existing rows.
targetbuffer	A value of the dwBuffer enumerated datatype identifying the target DataWindow buffer for the copied rows. For a list of valid values, see DWBuffer on page 372.

Return value

Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is NULL, the method returns NULL.

Usage

When you use the RowsCopy method, the status of the rows that are copied to the primary buffer is NewModified!. If you issue an update request, PocketBuilder sends SQL INSERT statements to the DBMS for the new rows.

When you use RowsCopy, data is not automatically retrieved for drop-down DataWindows in the target DataWindow or DataStore, as it is when you call InsertRow. You must explicitly call Retrieve for child DataWindows in the target.

When you use RowsCopy or RowsMove to populate another DataWindow, the copied data is not automatically processed by filters or sort criteria in effect on the target DataWindow. You might be required to call the Filter, GroupCalc, or Sort methods to properly process the data.

Uses for RowsCopy include:

- Making copies of one or more rows so that the users can create new rows based on existing data
- Printing a range of rows by copying selected rows to another DataWindow and printing the second DataWindow

Buffer manipulation and query mode
A DataWindow cannot be in query mode when you call the RowsCopy method.

Examples

This statement copies all the rows starting with the current row in dw_1 to the beginning of the primary buffer in dw_2:

```
dw_1.RowsCopy(dw_1.GetRow(), &
       dw_1.RowCount(), Primary!, dw_2, 1, Primary!)
```

This example copies all the rows starting with the current row in dw_1 to the beginning of the primary buffer in the drop-down DataWindow state_id in dw_3:

```
datawindowchild dwc
dw_3.GetChild("state_id", dwc)
dw_1.RowsCopy(dw_1.GetRow(), &
       dw_1.RowCount(), Primary!, dwc, 1, Primary!)
```

This example copies all the rows starting with the current row in dw_1 to the beginning of the primary buffer in the nested report d_employee:

```
datawindowchild dwc
dw_composite.GetChild("d_employee", dwc)
dw_1.RowsCopy(dw_1.GetRow(), &
       dw_1.RowCount(), Primary!, dwc, 1, Primary!)
```

See also

RowsDiscard
RowsMove

RowsDiscard

Description

Discards a range of rows in a DataWindow control. Once a row has been discarded using RowsDiscard, you cannot restore the row. You have to retrieve it again from the database.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

integer *dwcontrol*.**RowsDiscard** (long *startrow*, long *endrow*, DWBuffer *buffer*)

Argument	Description
dwcontrol	The reference to a DataWindow control or child DataWindow.
startrow	The number of the first row you want to discard.
endrow	The number of the last row you want to discard.

Argument	Description
buffer	A value of the dwBuffer enumerated datatype specifying the DataWindow buffer containing the rows to be discarded. For a list of valid values, see DWBuffer on page 372.

Return value

Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is NULL, the method returns NULL.

Usage

Use RowsDiscard when your application is finished with some of the rows in a DataWindow control and you do not want an update to affect the rows in the database. For example, you can discard rows in the delete buffer, which prevents the rows from being deleted when you call Update.

Use Reset to clear all the rows from a DataWindow control.

Examples

This statement discards all the rows in the delete buffer for dw_1. As a result if the application later calls dw_1.Update(), the DataWindow will not submit SQL DELETE statements to the DBMS for these rows:

```
dw_1.RowsDiscard(1, dw_1.DeletedCount(), Delete!)
```

See also

Reset
RowsCopy
RowsMove

RowsMove

Description

Clears a range of rows from one DataWindow control (or DataStore) and inserts them in another. Alternatively, RowsMove moves rows from one buffer to another within a single DataWindow control (or DataStore).

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

integer *dwcontrol*.**RowsMove** (long *startrow*, long *endrow*, DWBuffer *movebuffer*, datawindow *targetdw*, long *beforerow*, DWBuffer *targetbuffer*)

integer *dwcontrol*.**RowsMove** (long *startrow*, long *endrow*, DWBuffer *movebuffer*, datastore *targetdw*, long *beforerow*, DWBuffer *targetbuffer*)

integer *dwcontrol*.**RowsMove** (long *startrow*, long *endrow*, DWBuffer *movebuffer*, datawindowchild *targetdw*, long *beforerow*, DWBuffer *targetbuffer*)

Argument	Description
dwcontrol	The name of a DataWindow control, DataStore, or child DataWindow from which you want to move rows.
startrow	The number of the first row you want to move.
endrow	The number of the last row you want to move.
movebuffer	A value of the dwBuffer enumerated datatype identifying the DataWindow buffer from which you want to move the rows. For a list of valid values, see DWBuffer on page 372.
targetdw	The name of the DataWindow control or DataStore to which you want to move the rows. *Targetdw* can be the same DataWindow control (or DataStore) or a different DataWindow control (or DataStore), but it cannot be a child DataWindow.
beforerow	The number of the row before which you want to insert the moved rows. To insert after the last row, use any value that is greater than the number of existing rows.
targetbuffer	A value of the dwBuffer enumerated datatype identifying the target buffer for the rows. For a list of valid values, see DWBuffer on page 372.

Return value

Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is NULL, the method returns NULL.

Usage

When you use RowsMove, the rows have the status NewModified! in the target DataWindow.

If you move rows between buffers in a single DataWindow control or DataStore, PocketBuilder retains knowledge of where the rows came from, and their status is changed accordingly. For example, if you move unmodified rows from the primary buffer to the delete buffer, they are marked for deletion. If you move the rows back to the primary buffer, their status returns to NotModified!. Note, however, that if you move rows from one DataWindow control (or DataStore) to another and back again, the rows' status is NewModified! because they came from a different DataWindow.

When you use RowsMove, data is not automatically retrieved for drop-down DataWindows in the target DataWindow, as it is when you call InsertRow. You must explicitly call Retrieve for child DataWindows in the target.

When you use RowsCopy or RowsMove to populate another DataWindow, the copied data is not automatically processed by filters or sort criteria in effect on the target DataWindow. You might be required to call the Filter, GroupCalc, or Sort methods to properly process the data.

Uses for RowsMove include:

- Moving several rows from the primary buffer to the delete buffer, instead of deleting them one at a time
- Moving rows from the delete buffer to the primary buffer, to implement an Undo capability in your application

Buffer manipulation and query mode
A DataWindow cannot be in query mode when you call the RowsMove method.

Examples

This statement moves all the rows starting with the first row in the delete buffer for dw_1 to the primary buffer for dw_1, thereby *undeleting* these rows:

```
dw_1.RowsMove(1, dw_1.DeletedCount(), Delete!, &
       dw_1, 1, Primary!)
```

See also

RowsCopy
RowsDiscard

SaveAs

Description

Saves the contents of a DataWindow or DataStore in the format you specify.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

For syntax to save the contents of graphs in DataWindows and DataStores, see SaveAs on page 686.

Syntax

integer *dwcontrol*.**SaveAs** ({ string *filename*, saveastype *saveastype*, boolean *colheading* })

Argument	Description
dwcontrol	A reference to a DataWindow control, DataStore, or child DataWindow.
filename (optional)	A string whose value is the name of the file in which to save the contents. If you omit *filename* or specify an empty string (""), the DataWindow prompts for the filename. **Working with DataStore objects** If you are working with a DataStore, you must supply the *filename* argument.
saveastype (optional)	A value of the SaveAsType enumerated datatype specifying the format in which to save the contents of the DataWindow object. For a list of values, see SaveAsType on page 378.
colheading (optional)	A boolean value indicating whether you want to include the DataWindow's column headings at the beginning of the file. The default value is TRUE. This argument is used for the following formats: Clipboard, CSV, XLS, and TXT. For most other formats, column headings are always saved.

Return value

Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is NULL, SaveAs returns NULL.

Usage

If you do not specify any arguments for SaveAs, PocketBuilder displays the Save As dialog box. A drop-down list lets the user specify the format of the saved data.

If you use date formats in your report, 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 might need to reboot after the setting is changed.

When you save the contents of a DataWindow to a text file, double quotes are handled in a way that enables the ImportFile function to produce the same DataWindow when the text file is imported back into PocketBuilder. Any field that is enclosed in a pair of double quotes is wrapped with three pairs of double quotes in the saved text file. Double quotes at the beginning of a text field that have no matching double quotes at the end of the field are also replaced by three double quotes in the saved text file. However, a double quote elsewhere in the field is saved as one double quote.

Examples

This statement saves the contents of dw_History to the file G:\INVENTORY\EMPLOYEE.HIS. The saved file is in CSV format without column headings:

```
dw_History.SaveAs("G:\INVENTORY\EMPLOYEE.HIS", &
   CSV!, FALSE)
```

See also

ImportFile
Print
Update

SaveAsAscii

Description

Saves the contents of a DataWindow or DataStore into a standard ASCII text file.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

long *dwcontrol*.**SaveAsAscii** (string *filename* {, string *separatorcharacter* {,string *quotecharacter* {, string *lineending* } } })

Argument	Description
dwcontrol	A reference to a DataWindow control or DataStore.

Argument	Description
filename	A string whose value is the name of the file in which to save the contents.
separatorcharacter (optional)	A string whose value is the character to be used to delimit values. If you omit *separatorcharacter*, the default is a tab character.
quotecharacter (optional)	A string whose value is the character to be used to wrap values. If you omit *quotecharacter*, the default is double quote.
lineending (optional)	A string whose value is placed at the end of each line. If you omit *lineending*, the default is a carriage return plus a newline character (~r~n).

Return value

Returns 1 if it succeeds and -1 if an error occurs.

Usage

SaveAsAscii is a cross between the SaveAs (Text!) function and the SaveAs (HTMLTable!) function with additional arguments. It mirrors more closely what the user sees on the screen. Arguments allow the user to control how contents are separated and delimited in the ASCII file.

PocketBuilder assigns a cell for each DataWindow object (which can include computed columns and group totals). If a cell is empty, PocketBuilder puts the *quotecharacter* between the *separatorcharacter* in the output file.

Examples

This statement saves the contents of dw_Quarter to the file H:\Q2\RESULTS.TXT. The saved file is ASCII with the ampersand (&) as the separator character, and single quotes (') as the characters used to wrap values. A new line (~r~n) is automatically inserted at each line ending. Computed columns are included with the saved information:

```
dw_Quarter.SaveAsAscii("H:\Q2\RESULTS.TXT","&","'")
```

See also

SaveAs

Scroll

Description

Scrolls the edit control of a DataWindow a specified number of lines up or down.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

long *dwcontrol*.**Scroll** (long *number*)

Argument	Description
dwcontrol	A reference to a DataWindow control. Scroll affects the edit control of the DataWindow.
number	A value specifying the direction and number of lines you want to scroll. To scroll down, use a positive value. To scroll up, use a negative value.

Return value

Scroll returns the line number of the first visible line in *dwcontrol* 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 will scroll 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

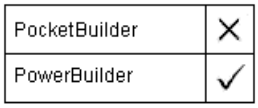
The following related methods implement scrolling in a DataWindow:

ScrollNextPage
ScrollNextRow
ScrollPriorPage
ScrollPriorPage
ScrollToRow

ScrollFirstPage

Description

Scrolls a Web DataWindow control to the first page, displaying the result set's first group of rows in the Web page. (A page is the number of rows that are displayed in the DataWindow control at one time.) ScrollFirstPage changes the current row, but not the current column.



Syntax

Web DataWindow client control

number *dwcontrol*.**ScrollFirstPage** ( )

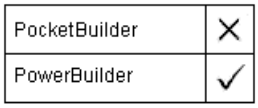
Return value

Returns 1 if it succeeds and -1 if an error occurs. If *dwcontrol* is NULL, the method returns NULL.

ScrollLastPage

Description

Scrolls a Web DataWindow control to the last page, displaying the result set's last group of rows in the Web page. (A page is the number of rows that are displayed in the DataWindow control at one time.) ScrollLastPage changes the current row, but not the current column.



Syntax

Web DataWindow client control

number *dwcontrol*.**ScrollLastPage** ( )

Return value

Returns 1 if it succeeds and -1 if an error occurs. If *dwcontrol* is NULL, the method returns NULL.

ScrollNextPage

Scrolls to the next page in a DataWindow.

To scroll	Use
To the next group of rows in a DataWindow (when the DataWindow does not have the RichTextEdit presentation style)	Syntax 1
A RichTextEdit DataWindow to view the next page within the document (PowerBuilder only)	Syntax 2

Syntax 1 For DataWindow controls and child DataWindows

Description

Scrolls a DataWindow control forward one page, displaying the next group of rows in the DataWindow's display area. (A page is the number of rows that can be displayed in the DataWindow control at one time.) ScrollNextPage changes the current row, but not the current column.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

long *dwcontrol*.**ScrollNextPage** ( )

Argument	Description
dwcontrol	A reference to a DataWindow control or child DataWindow

Return value

Returns the number of the row displayed at the top of the DataWindow control when the scroll finishes or tries to scroll past the last row. ScrollNextPage returns 1 with nested or composite reports and child DataWindows since, in these cases, the current row cannot be changed. ScrollNextPage returns -1 if an error occurs.

If *dwcontrol* is NULL, the method returns NULL.

Usage

ScrollNextPage does not highlight the current row. Use SelectRow to let the user know what row is current.

For an example that uses RowCount and Describe to check whether the user has scrolled to the last page, see RowCount.

Events ScrollNextPage can trigger these events:

ItemChanged
ItemError
ItemFocusChanged

RowFocusChanged
RowFocusChanging

Examples

This statement scrolls dw_employee forward one page:

```
dw_employee.ScrollNextPage()
```

See also

Scroll
ScrollFirstPage
ScrollLastPage
ScrollNextRow
ScrollPriorPage
ScrollPriorRow
ScrollToRow
SelectRow

Syntax 2 For RichTextEdit DataWindows

Description

Scrolls to the next page of the document in a RichTextEdit DataWindow.

PocketBuilder	✗
PowerBuilder	✓

Syntax

PowerBuilder DataWindow control

integer *rtedwname*.**ScrollNextPage** ( )

Return value

Returns 1 if it succeeds and -1 if an error occurs. If *rtedwname* is NULL, in PowerBuilder and JavaScript the method returns NULL.

ScrollNextRow

Scrolls to the next row in a DataWindow control.

To scroll	Use
To the next row in a DataWindow, making the row current (when the DataWindow does not have the RichTextEdit presentation style)	Syntax 1
To the next instance of a document associated with a row in a RichTextEdit DataWindow (PowerBuilder only)	Syntax 2

Syntax 1 For DataWindow controls and child DataWindows

Description

Scrolls a DataWindow control to the next row (forward one row). ScrollNextRow changes the current row, but not the current column.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

long *dwcontrol*.**ScrollNextRow** ( )

Argument	Description
dwcontrol	A reference to a DataWindow control or child DataWindow

Return value

Returns the number of the row displayed at the top of the DataWindow control when the scroll finishes or tries to scroll past the last row. ScrollNextRow returns -1 if an error occurs. If *dwcontrol* is NULL, the method returns NULL.

Usage

After you call ScrollNextRow, the row after the current row becomes the new current row. If that row is already visible, the displayed rows do not change. If it is not visible, the displayed rows move up to display the row.

ScrollNextRow does not highlight the row. Use SelectRow to let the user know what row is current.

Events ScrollNextRow triggers these events in the order shown:

RowFocusChanging
RowFocusChanged
ItemFocusChanged
ScrollVertical

You should not use ScrollNextRow in the ScrollVertical event. Doing so causes this series of events to be triggered repeatedly until the last row in the DataWindow is reached.

Examples

This statement scrolls dw_employee to the next row:

```
dw_employee.ScrollNextRow()
```

See also

Scroll
ScrollNextPage
ScrollPriorPage
ScrollPriorRow
ScrollToRow
SelectRow

Syntax 2 For RichTextEdit DataWindows

Description

Scrolls to the next instance of the document in a RichTextEdit DataWindow.

PocketBuilder	✗
PowerBuilder	✓

Syntax

PowerBuilder DataWindow control

integer *rtename*.**ScrollNextRow** ( )

Return value

Returns 1 if it succeeds and -1 if an error occurs.

ScrollPriorPage

Scrolls to the prior page in a DataWindow control.

To scroll	Use
To the prior group of rows in a DataWindow (when the DataWindow does not have the RichTextEdit presentation style)	Syntax 1
A RichTextEdit DataWindow to view the prior page within the document (PowerBuilder only)	Syntax 2

Syntax 1 For DataWindow controls and child DataWindows

Description

Scrolls a DataWindow control backward one page, displaying another group of rows in the DataWindow's display area. (A page is the number of rows that can be displayed in the DataWindow control at one time.) ScrollPriorPage changes the current row but not the current column.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

long *dwcontrol*.**ScrollPriorPage** ( )

Argument	Description
dwcontrol	The name of the DataWindow control or child DataWindow you want to have scroll to the prior page

Return value

Returns the number of the row displayed at the top of the DataWindow control when the scroll finishes or tries to scroll past the first row. ScrollPriorPage returns -1 if an error occurs. If *dwcontrol* is NULL, the method returns NULL.

Usage

ScrollPriorPage does not highlight the current row. Use SelectRow to let the user know what row is current.

Web DataWindow Calling ScrollNextPage causes the page to be reloaded with another set of rows from the result set.

If the DataWindow object has retrieval arguments, they must be specified in the HTMLGen.SelfLinkArgs property. For more information, see the HTMLGen.property, the Retrieve method, and the *DataWindow Programmer's Guide*.

All methods that reload the page perform an AcceptText before sending data back to the server. If DeleteRow fails (returns -1), this means that pending data changes were not accepted and nothing was sent back to the server. In this situation the ItemError event occurs.

Events ScrollPriorPage can trigger these events:

ItemChanged
ItemError
ItemFocusChanged
RowFocusChanged
RowFocusChanging

Examples

This statement scrolls dw_employee backward one page:

```
dw_employee.ScrollPriorPage()
```

See also

Scroll
ScrollFirstPage
ScrollLastPage
ScrollNextPage
ScrollNextRow
ScrollPriorRow
ScrollToRow
SelectRow

Syntax 2 For RichTextEdit DataWindows

Description

Scrolls to the prior page of the document in a RichTextEdit DataWindow.

PocketBuilder	✗
PowerBuilder	✓

Syntax

PowerBuilder DataWindow control

integer *rtename*.**ScrollPriorPage** ( )

Return value

Returns 1 if it succeeds and -1 if an error occurs.

ScrollPriorRow

Scrolls to the prior row in a DataWindow control.

To scroll	Use
To the prior row in a DataWindow, making the row current (when the DataWindow does not have the RichTextEdit presentation style)	Syntax 1
To the prior instance of a document associated with a row in a RichTextEdit control or RichTextEdit DataWindow	Syntax 2

Syntax 1 For DataWindow controls and child DataWindows

Description

Scrolls a DataWindow control backward one row. ScrollPriorRow changes the current row but not the current column.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

long *dwcontrol*.**ScrollPriorRow** ( )

Argument	Description
dwcontrol	A reference to a DataWindow or child DataWindow

Return value

Returns the number of the row displayed at the top of the DataWindow control when the scroll finishes or tries to scroll past the first row. ScrollPriorRow returns -1 if an error occurs. If *dwcontrol* is NULL, the method returns NULL.

Usage

After you call ScrollPriorRow, the row before the current row becomes the new current row. If that row is already visible, the displayed rows do not change. If it is not visible, the displayed rows move down to display the row.

ScrollPriorRow does not highlight the row. Use SelectRow to let the user know what row is current.

Events ScrollPriorRow triggers these events in the order shown:

RowFocusChanging
RowFocusChanged
ItemFocusChanged
ScrollVertical

You should not use ScrollPriorRow in the ScrollVertical event. Doing so causes this series of events to be triggered repeatedly until the first row in the DataWindow is reached.

Examples

This statement scrolls dw_employee to the prior row:

```
dw_employee.ScrollPriorRow()
```

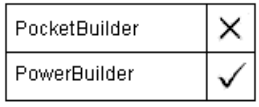
See also

Scroll
ScrollNextPage
ScrollNextRow
ScrollPriorPage
ScrollToRow
SelectRow

Syntax 2 For RichTextEdit DataWindows

Description

Scrolls to the prior instance of the document in a RichTextEdit DataWindow.



Syntax

PowerBuilder DataWindow control

integer *rtename*.**ScrollPriorRow** ( )

Return value

Returns 1 if it succeeds and -1 if an error occurs.

ScrollToRow

Description

Scrolls a DataWindow control to the specified row. ScrollToRow changes the current row but not the current column.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

integer *dwcontrol*.**ScrollToRow** (long *row*)

Argument	Description
dwcontrol	A reference to a DataWindow control or child DataWindow.
row	A value identifying the row to which you want to scroll. If *row* is 0, ScrollToRow scrolls to the first row. If *row* is greater than the last row number, it scrolls to the last row.

Return value

Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is NULL, the method returns NULL.

Usage

After you call ScrollToRow, the specified row becomes the new current row. If that row is already visible, the displayed rows do not change. If it is not visible, the displayed rows change to display the row.

ScrollToRow does not highlight the row. Use SelectRow to let the user know what row is current.

Events ScrollToRow can trigger these events:

ItemChanged
ItemError
ItemFocusChanged
RowFocusChanged

Examples

This statement scrolls to row 10 and makes it current in the DataWindow control dw_employee:

```
dw_employee.ScrollToRow(10)
```

See also

Scroll
ScrollNextPage
ScrollNextRow
ScrollPriorPage
ScrollPriorRow
SelectRow

SelectedLength

Description

Determines the total number of characters in the selected text in an edit control, including spaces and line endings.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

long *dwcontrol*.**SelectedLength** ( )

Argument	Description
dwcontrol	A reference to a DataWindow control. SelectedLength reports the length of the selected text in the edit control over the current row and column.

Return value

Returns the length of the selected text in *dwcontrol*. If no text is selected, SelectedLength returns 0. If an error occurs, it returns -1. If *dwcontrol* is NULL, the method returns NULL.

Usage

The characters that make up a line ending, produced by typing CTRL+ENTER or ENTER, are different on different platforms. On Windows, they are a carriage return plus a line feed and equals two characters when calculating the length. On other platforms, a line ending can be a single character. A line that wraps has no line-ending character.

Using with other PocketBuilder controls
For use with other controls, see SelectedLength in the *PowerScript Reference*.

Examples

If the selected text in the DataWindow dw_Contact is John Smith, then this example sets the variable to 10, the number of selected characters:

```
integer li_length

li_length = dw_Contact.SelectedLength()
```

See also

SelectedLine
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	✓

Syntax

long *dwcontrol*.**SelectedLine** ( )

Argument	Description
dwcontrol	A reference to a DataWindow control. It reports the line number in the edit control over the current row and column.

Return value

Returns the number of the line containing the insertion point in *dwcontrol*. If an error occurs, SelectedLine returns -1. If *dwcontrol* is NULL, SelectedLine returns NULL.

Usage

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.

Using with other PocketBuilder controls
For use with other controls, see SelectedLine in the *PowerScript Reference*.

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

Position
SelectedText
TextLine

SelectedStart

Description

Reports the position of the first selected character in the edit control.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

long *dwcontrol*.**SelectedStart** ( )

Argument	Description
dwcontrol	A reference to a DataWindow control. It reports the starting position in the edit control over the current row and column.

Return value

Returns the starting position of the selected text in *dwcontrol*. 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 *dwcontrol* is NULL, the method returns NULL.

Usage

SelectedStart counts from the start of the text and includes spaces and line endings.

Using with other PocketBuilder controls
For use with other controls, see SelectedStart in the *PowerScript Reference*.

Examples

If the edit control for the DataWindow control dw_rpt contains Closed for Vacation July 3 to July 10, and Vacation is selected, then this example sets the variable to 12 (the position of the first character in Vacation):

```
integer li_Start
li_Start = dw_rpt.SelectedStart()
```

See also

Position
SelectedLength
SelectedLine

SelectedText

Description

Obtains the selected text in the edit control of a DataWindow control.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

string *dwcontrol*.**SelectedText** ( )

Argument	Description
dwcontrol	A reference to a DataWindow control. The method reports the selected text in the edit control over the current row and column.

Return value

Returns the selected text in *dwcontrol*. If there is no selected text or if an error occurs, SelectedText returns the empty string (""). If *dwcontrol* is NULL, the method returns NULL.

Usage

Using with other PocketBuilder controls
For use with other controls, see SelectedText in the *PowerScript Reference*.

Examples

If the text in the edit control of the DataWindow dw_rpt is James B. Smith and James B. is selected, these statements set the value of the string variable to James B:

```
string ls_emp_fname
ls_emp_fname = dw_rpt.SelectedText()
```

See also

SelectText

SelectRow

Description

Highlights or removes highlights from rows in a DataWindow control or DataStore. You can select all rows or a single row. SelectRow does not affect which row is current. It does not select rows in the database.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

integer *dwcontrol*.**SelectRow** (long *row*, boolean *select*)

Argument	Description
dwcontrol	A reference to a DataWindow control, DataStore, or child DataWindow.
row	A value identifying the row you want to select or deselect. Specify 0 to select or deselect all rows.
select	A boolean value that determines whether the row is selected or not selected: • TRUE — Select the row(s) so that they are highlighted. • FALSE — Deselect the row(s) so that they are not highlighted.

Return value

Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is NULL, the method returns NULL. If there is no DataWindow object assigned to the DataWindow control or DataStore, the method returns 1.

Usage

If a row is already selected and you specify that it be selected (*boolean* is TRUE), it remains selected. If a row is not selected and you specify that it not be selected (*boolean* is FALSE), it remains unselected.

Examples

This statement selects the fifteenth row in dw_employee:

```
dw_employee.SelectRow(15, TRUE)
```

As the script for a DataWindow's Clicked event, this example removes highlighting from all rows and then highlights the row the user clicked.

Row is an argument passed to the event script:

```
This.SelectRow(0, FALSE)
This.SelectRow(row, TRUE)
```

SelectText

Selects text in an edit control.

To select text in	Use
A DataWindow when the DataWindow does not have the RichTextEdit presentation style	Syntax 1
A DataWindow whose object has the RichTextEdit presentation style (PowerBuilder only)	Syntax 2

Syntax 1 For DataWindows with standard edit styles

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	✓

Syntax

long *dwcontrol*.**SelectText** (long *start*, long *length*)

Argument	Description
dwcontrol	A reference to a DataWindow control.
start	A numeric value specifying the position at which you want to start the selection.
length	A numeric value specifying the number of characters you want to select. If *length* is 0, no text is selected but SelectText moves the insertion point to the location specified in *start*.

Return value

Returns the number of characters selected. If an error occurs, SelectText returns -1. If any argument's value is NULL, the method 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.

To select text in a DataWindow with the RichTextEdit presentation style, use Syntax 2.

Using with other PocketBuilder controls
For use with other controls, see SelectText in the *PowerScript Reference*.

Examples

This statement sets the insertion point at the end of the text in the DataWindow edit control:

```
dw_1.SelectText(dw_1.GetText(), 0)
```

This statement selects the entire contents of the DataWindow edit control:

```
dw_1.SelectText(1, Len(dw_1.GetText()))
```

The rest of these examples assume the DataWindow edit control contains Boston Street.

The following statement selects the string ost and returns 3:

```
dw_1.SelectText(2, 3)
```

The next statement selects the string oston Street and returns 12:

```
dw_1.SelectText(2, Len(dw_1.GetText()))
```

These statements select the string Bos, returns 3, and sets the focus to the DataWindow control so that Bos is highlighted:

```
dw_1.SelectText(1, 3)
dw_1.SetFocus()
```

See also

Position
SelectedText
TextLine

Syntax 2 For RichTextEdit DataWindows

Description

Selects text beginning and ending at a line and character position in a RichText DataWindow.

PocketBuilder	✗
PowerBuilder	✓

Syntax

PowerBuilder

long *rtedwcontrol*.**SelectText** (long *fromline*, long *fromchar*, long *toline*, long *tochar* { band *band* })

Return value

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 feed (LF), and end-of-file (EOF).

PocketBuilder	✗
PowerBuilder	✓

Syntax

PowerBuilder DataWindow control

integer *rtename*.**SelectTextAll** (band *band*)

Return value

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	✓

Syntax

PowerBuilder DataWindow control

integer *rtename*.**SelectTextLine** ( )

Return value

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	✓

Syntax

PowerBuilder DataWindow control

integer *rtename*.**SelectTextWord** ( )

Return value

Returns the number of characters selected if it succeeds and -1 if a word cannot be selected or an error occurs.

SetAction

Description

Accepts action and context information about user interaction with the Web DataWindow client control in a Web browser so that generated HTML reflects any requested changes.

PocketBuilder	✗
PowerBuilder	✓

Syntax

Web DataWindow server component

integer *dwcomponent*.**SetAction** (string *action*, string *context*)

Return value

Returns 1 if it succeeds and a negative value if an error occurs.

SetActionCode

Description

Sets the action code for an event in a DataWindow control. The action code determines the action that PowerBuilder takes following the event. The default action code is 0.

PocketBuilder	✗
PowerBuilder	✓

Where to use SetActionCode
SetActionCode is obsolete. To return a value, include a RETURN statement in the event script using the return codes documented for that event.

Syntax

PowerBuilder DataWindow control or child DataWindow

integer *dwcontrol*.**SetActionCode** (long *code*)

Return value

Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is NULL, SetActionCode returns NULL.

SetBorderStyle

Description

Sets the border style of a column in a DataWindow control or DataStore.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

integer *dwcontrol*.**SetBorderStyle** (integer *column*, border *borderstyle*)

integer *dwcontrol*.**SetBorderStyle** (string *column*, border *borderstyle*)

Argument	Description
dwcontrol	A reference to a DataWindow control, DataStore, or child DataWindow.

Argument	Description
column	The column in which you want to change the border style. *Column* can be a column number or a column name.
borderstyle	A value of the Border enumerated datatype identifying the border style you want to use for the column. For a list of valid values, see Border on page 369.

Return value

Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is NULL, the method returns NULL.

Examples

This example checks the border of column 2 in dw_emp and, if there is no border, gives it a shadow box border:

```
Border B3
B3 = dw_emp.GetBorderStyle(2)
IF B3 = NoBorder! THEN &
       dw_emp.SetBorderStyle(2, ShadowBox!)
```

See also

GetBorderStyle

SetBrowser

Description

Specifies the Web browser for which you want to generate optimized HTML.

PocketBuilder	✗
PowerBuilder	✓

Syntax

Web DataWindow server component

string *dwcomponent*.**SetBrowser** (string *browsername*)

Return value

Returns an empty string if successful and the syntax error message from the Modify method if it fails.

SetChanges

Description

Applies changes captured with GetChanges to a DataWindow or DataStore. This method is used primarily in distributed applications.

PocketBuilder	✗
PowerBuilder	✓

Syntax

PowerBuilder DataWindow control or DataStore object

long *dwcontrol*.**SetChanges** (blob *changeblob* {, dwConflictResolution *resolution }*)

Return value

Returns 1 for success and -1 for failure.

If any argument's value is null, in PowerBuilder and JavaScript the method returns null.

SetColumn

Description

Sets the current column in a DataWindow control or DataStore.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

integer *dwcontrol*.**SetColumn** (string *column*)

integer *dwcontrol*.**SetColumn** (integer *column*)

Argument	Description
dwcontrol	A reference to a DataWindow control, DataStore, or child DataWindow.
column	The column you want to make current. *Column* can be a column number or a column name.

Return value

Returns 1 if it succeeds and -1 if an error occurs. If *column* is less than 1 or greater than the number of columns, SetColumn fails. If any argument's value is NULL, the method returns NULL.

Usage

SetColumn moves the cursor to the current column but does not scroll the DataWindow control.

Only an editable column can be current. (A column is editable when its tab order value is greater than 0.) Do not try to set a noneditable column as the current column.

Using with other PocketBuilder controls
For use with ListView controls, see SetColumn in the *PowerScript Reference*.

Events SetColumn can trigger these events:

- ItemChanged
- ItemError
- ItemFocusChanged

Avoiding infinite loops
Never call SetColumn in the ItemChanged, ItemError, or ItemFocusChanged event. Because SetColumn can trigger these events, such a recursive call can cause a stack fault.

Examples

This statement makes the 15th column in dw_Employee the current column:

```
dw_Employee.SetColumn(15)
```

See also

GetColumn
GetRow
SetRow

SetColumnLink

Description

Specifies information used for constructing hyperlinks for data in a column in generated HTML.

PocketBuilder	✗
PowerBuilder	✓

Syntax

Web DataWindow PSWebDataWindowClass and server component

string *dwcomponent*.**SetColumnLink** (string *columnname*, string *link*, string *linkargs*, string *linktarget*)

Return value

Returns an empty string if successful and the syntax error message from the Modify method if it fails.

SetDetailHeight

Description

Sets the height of each row in the specified range to the specified value.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

integer *dwcontrol*.**SetDetailHeight** (long *startrow*, long *endrow* , long *height*)

Argument	Description
dwcontrol	A reference to a DataWindow control or DataStore for which you want to set the height of one or more rows in the detail area
startrow	The first row in the range of rows for which you want to set the height
endrow	The last row in the range of rows for which you want to set the height
height	The height of the detail area for the specified rows in the units specified for the DataWindow object

Return value

Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is NULL, the method returns NULL.

Usage

Call SetDetailHeight in a script to vary the amount of space assigned to rows in a DataWindow control or DataStore. You cannot specifically set the height for different rows when you define a DataWindow object in the DataWindow painter, although you can turn on the Autosize Height property for the detail band so that the height of each row is determined by the data.

You can set the detail height of one or more rows to zero, which hides them from view.

Examples

This statement sets the height of rows 2 and 3 to 500:

```
dw_1.SetDetailHeight(2, 3, 500)
```

This script retrieves rows for a DropDownDataWindow associated with the Company_Name column. It then hides rows 2 and 3 of the DropDownDataWindow by setting their detail height to 0:

```
DataWindowChild dwc;
integer rtncode;

rtncode = dw_1.GetChild("company_name", dwc)
IF rtncode < 0 THEN HALT

dwc.SetTransObject(SQLCA)
```

```
dwc.Retrieve( )
dwc.SetDetailHeight(2, 3, 0)
```

SetDWObject

Description

Specifies the DataWindow library and object that the Web DataWindow server component will use for generating HTML.

PocketBuilder	✗
PowerBuilder	✓

Syntax

Web DataWindow server component

int *dwcomponent*.**SetDWObject** (string *sourcefile*, string *dwobjectname*)

int *dwcomponent*.**SetDWObjectEx** (string *dwobjectname*)

Return value

Returns 1 if it succeeds and -1 if an error occurs.

SetFilter

Description

Specifies filter criteria for a DataWindow control or DataStore.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

integer *dwcontrol*.**SetFilter** (string *format*)

Argument	Description
dwcontrol	The name of the DataWindow control, DataStore, or child DataWindow in which you want to define the filter.
format	A string whose value is a boolean expression that you want to use as the filter criteria. The expression includes column names or numbers. A column number must be preceded by a pound sign (#). If *format* is NULL, PocketBuilder prompts you to enter a filter.

Return value

Returns 1 if it succeeds and -1 if an error occurs. The return value is usually not used.

Usage

A DataWindow object can have filter criteria specified as part of its definition. After data is retrieved, rows that do not meet the criteria are immediately transferred from the primary buffer to the filter buffer.

The SetFilter method replaces the existing filter criteria—if any are defined for the DataWindow object—with a new set of criteria. Call the Filter method to apply the filter criteria and transfer rows that do not meet the filter criteria to the filter buffer.

The filter expression consists of columns, relational operators, and values against which column values are compared. Boolean expressions can be connected with logical operators AND and OR. You can also use NOT, the negation operator. Use parentheses to control the order of evaluation.

Sample expressions are:

```
item_id > 5
NOT item_id = 5
(NOT item_id = 5) AND customer > "Mabson"
item_id > 5 AND customer = "Smith"
#1 > 5 AND #2 = "Smith"
```

The filter expression is a string and does not contain variables. However, you can build the string in your script using the values of script variables. Within the filter string, string constants must be enclosed in quotation marks (see the examples).

If the filter expression contains numbers, the DataWindow expects the numbers in U.S. format. In PocketBuilder, be aware that the String function formats numbers using the current system settings. If you use it to build the filter expression, specify a display format that produces U.S. notation.

Removing a filter
To remove a filter, call SetFilter with the empty string ("") for *format* and then call Filter. The rows in the filter buffer will be restored to the primary buffer and positioned after the rows that already exist in the primary buffer.

To let users specify their own filter expression for a DataWindow control, you can pass a null string to the SetFilter method. PocketBuilder displays its Specify Filter dialog box with the filter expression blank. Then you can call Filter to apply the user's filter expression to the DataWindow. You cannot pass a null string to the SetFilter method for a DataStore object.

Examples

This statement defines the filter expression for dw_Employee as the value of format1:

```
dw_Employee.SetFilter(format1)
```

The following statements define a filter expression and set it as the filter for dw_Employee. With this filter, only those rows in which the cust_qty column exceeds 100 and the cust_code column exceeds 30 are displayed. The final statement calls Filter to apply the filter:

```
string DWfilter2
DWfilter2 = "cust_qty > 100 and cust_code >30"
dw_Employee.SetFilter(DWfilter2)
dw_Employee.Filter( )
```

The following statements define a filter so that emp_state of dw_Employee displays only if it is equal to the value of var1 (in this case ME for Maine). The filter expression passed to SetFilter is emp_state = ME:

```
string Var1
Var1 = "ME"
dw_Employee.SetFilter("emp_state = '"+ var1 +" '")
```

The following statements define a filter so that column 1 must equal the value in min_qty and column 2 must equal the value in max_qty to pass the filter. The resulting filter expression is:

```
#1=100 and #2=1000
```

The sample code is:

```
integer max_qty, min_qty
min_qty = 100
max_qty = 1000
dw_inv.SetFilter("#1="+ String( min_qty) &
       + " and #2=" + String(max_qty))
```

The following example sets the filter expression to null, which causes PocketBuilder to display the Specify Filter dialog box. Then it calls Filter, which applies the filter expression the user specified:

```
string null_str
SetNull(null_str)
dw_main.SetFilter(null_str)
dw_main.Filter()
```

See also

Filter

SetFormat

Description

Specifies a display format for a column in a DataWindow control or DataStore.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

integer *dwcontrol*.**SetFormat** (string *column*, string *format*)

integer *dwcontrol*.**SetFormat** (integer *column*, string *format*)

Argument	Description
dwcontrol	A reference to a DataWindow control, DataStore, or child DataWindow.
column	The column for which you are specifying the display format. *Column* can be a column number or a column name.
format	A string whose value is the display format for the DataWindow column.

Return value

Returns 1 if it succeeds and -1 if an error occurs. The return value is usually not used. If any argument's value is NULL, the method returns NULL.

Usage

For information on valid display formats for different datatypes, see the *Users Guide*.

If you are specifying the display format for a number, the format must use U.S. notation. For example, comma (,) represents the thousands delimiter and period (.) represents the decimal place. During execution, the locally correct symbols will be displayed.

An EditMask edit style supersedes any display format applied to the column. When the column has an EditMask edit style, calling SetFormat has no effect.

Examples

These statements define the display format for column 15 of dw_employee to the contents of format1:

```
string format1
format1 = "$#,##0.00"
dw_employee.SetFormat(15, format1)
```

See also

GetFormat

SetFullState

Description

Applies the contents of a DataWindow blob retrieved by GetFullState to a DataWindow or DataStore. This method is used primarily in distributed applications.

PocketBuilder	✗
PowerBuilder	✓

Syntax

PowerBuilder DataWindow control or DataStore object

long *dwcontrol*.**SetFullState** (blob *dwasblob*)

Return value

Returns 1 for success and -1 for failure. If any argument's value is null, the method returns null.

SetHTMLAction

Description

Accepts action and context information about user interaction with the Web DataWindow client control in a Web browser so that newly generated HTML can reflect any requested changes.

PocketBuilder	✗
PowerBuilder	✓

Syntax

PowerBuilder

integer *dwcontrol*.**SetHTMLAction** (string *action*, string *context*)

Return value

Returns 1 if it succeeds and a negative value if an error occurs.

SetHTMLObjectName

Description

Specifies a name for the Web DataWindow client control.

PocketBuilder	✗
PowerBuilder	✓

Syntax

Web DataWindow server component

string *dwcomponent*.**SetHTMLObjectName** (string *objectname*)

Return value

Returns an empty string if successful and the syntax error message from the Modify method if it fails.

SetItem

Description

Sets the value of a row and column in a DataWindow control or DataStore to the specified value.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

integer *dwcontrol*.**SetItem** (long *row*, integer *column*, any *value*)

integer *dwcontrol*.**SetItem** (long *row*, string *column*, any *value*)

Argument	Description
dwcontrol	The name of the DataWindow control, DataStore, or child DataWindow in which you want to set a specific row and column to a value.
row	The row location of the data.
column	The column location of the data. *Column* can be a column number or a column name. The column number is the number of the column as it is listed in the Column Specification view of the DataWindow painter—not necessarily the number of the column in the Design view.
value	The value to which you want to set the data at the row and column location. The datatype of the value must be the same datatype as the column.

Return value

Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is NULL, the method returns NULL.

Usage

SetItem sets a value in a DataWindow buffer. It does not affect the value currently in the edit control over the current row and column, which is the data the user has changed or might change. The value in the edit control does not become the value of the DataWindow item until it is validated and accepted (see AcceptText). In a script, you can change the value in the edit control with the SetText method.

You can use SetItem when you want to set the value of an item in a DataWindow control or DataStore that has script as the source.

You can also use SetItem to set the value of an item when the data the user entered is not valid. When you use a return code that rejects the data the user entered but allows the focus to change (return code of 2 in the script of the ItemChanged event or return code of 3 in the ItemError event), you can call SetItem to put valid data in the row and column.

Using SetItem to correct user input
If PocketBuilder cannot properly convert the string the user entered, you must include statements in the script for the ItemChanged or ItemError event to convert the data and use SetItem with the converted data. For example, if the user enters a number with commas and a dollar sign (for example, $1,000), PocketBuilder is unable to convert the string to a number and you must convert it in the script.

If you use SetItem to set a row and column to a value other than the value the user entered, you can use SetText to assign the new value to the edit control so that the user sees the current value.

Using with other PocketBuilder controls
For use with ListView and TreeView controls, see SetItem in the *PowerScript Reference*.

Examples

This statement sets the value of row 3 of the column named hire_date of the DataWindow control dw_order to 1993-06-07:

```
dw_order.SetItem(3, "hire_date", 1993-06-07)
```

When a user starts to edit a numeric column and leaves it without entering any data, PocketBuilder tries to assign an empty string to the column. This fails the datatype validation test. In this example, code in the ItemError event sets the column's value to NULL and allows the focus to change.

This example assumes that the datatype of column 2 is numeric. If it is date, time, or datetime, replace the first line (integer null_num) with a declaration of the appropriate datatype:

```
integer null_num //to contain null value

SetNull(null_num)

// Special processing for column 2
IF dwo.ID = 2 THEN
```

```
        // If user entered nothing (""), set to null
        IF data = "" THEN
            This.SetItem(row, dwo.ID, null_num)
            RETURN 2
        END IF
END IF
```

The following example is a script for a DataWindow's ItemError event. If the user specifies characters other than digits for a numeric column, the data will fail the datatype validation test. You can include code to strip out characters such as commas and dollar signs and use SetItem to assign the now valid numeric value to the column. The return code of 3 causes the data in the edit control to be rejected because the script has provided a valid value:

```
string snum, c
integer cnt

// Extract the digits from the user's data
FOR cnt = 1 to Len(data)
       c = Mid(data, cnt, 1) // Get character
       IF IsNumber(c) THEN snum = snum + c
NEXT
This.SetItem(row, dwo.ID, Long(snum))
RETURN 3
```

See also

GetItemDate
GetItemDateTime
GetItemNumber
GetItemString
GetItemTime
GetText
SetText

SetItemDate

Description

Sets the value of a row and column in a DataWindow control to the specified value.

PocketBuilder	✗
PowerBuilder	✓

Syntax

Web DataWindow PSWebDataWindowClass

number *dwcontrol*.**SetItemDate** (number *row*, string *column*, Date *value*)

number *dwcontrol*.**SetItemDate** (number *row*, number *column*, Date *value*)

Web DataWindow server component

short *dwcontrol*.**SetItemDate** (long *row*, string *column*, string *value*)

short *dwcontrol*.**SetItemDateByColNum** (long *row*, short *column*, string *value*)

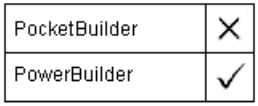
Return value

Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is NULL, the method returns NULL.

SetItemDateTime

Description

Sets the value of a row and column in a DataWindow control to the specified value.



Syntax

Web DataWindow PSWebDataWindowClass

number *dwcontrol*.**SetItemDateTime** (number *row*, string *column*, Date *value*)

number *dwcontrol*.**SetItemDateTime** (number *row*, number *column*, Date *value*)

Web DataWindow server component

short *dwcontrol*.**SetItemDateTime** (long *row*, string *column*, string *value*)

short *dwcontrol*.**SetItemDateTimeByColNum** (long *row*, short *column*, string *value*)

Return value

Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is NULL, the method returns NULL.

SetItemNumber

Description

Sets the value of a row and column in a DataWindow control to the specified value.

PocketBuilder	✗
PowerBuilder	✓

Syntax

Web DataWindow server component

short *dwcontrol*.**SetItemNumber** (long *row*, string *column*, double *value*)

short *dwcontrol*.**SetItemNumberByColNum** (long *row*, short *column*, double *value*)

Return value

Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is NULL, the method returns NULL.

SetItemStatus

Description

Changes the modification status of a row or a column within a row. The modification status determines the type of SQL statement the Update method will generate for the row.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

integer *dwcontrol*.**SetItemStatus** (long *row*, integer *column*, dwbuffer *dwbuffer*, dwitemstatus *status*)

integer *dwcontrol*.**SetItemStatus** (long *row*, string *column*, dwbuffer *dwbuffer*, dwitemstatus *status*)

Argument	Description
dwcontrol	A reference to a DataWindow control, DataStore, or child DataWindow.
row	The row location in which you want to set the status.
column	The column location in which you want to set the status. *Column* can be a column number or a column name. To set the status for the row, enter 0 for *column*.

Argument	Description
dwbuffer	A value identifying the DataWindow buffer that contains the row. For a list of valid values, see DWBuffer on page 372.
status	A value of the dwItemStatus enumerated datatype specifying the new status. For a list of valid values, see DWItemStatus on page 373.

Return value

Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is NULL, the method returns NULL.

Usage

How statuses are set There are four DataWindow item statuses, two of which apply only to rows:

Table 9-5: Possible statuses for DataWindow items

Status	Applies to
New!	Rows
NewModified!	Rows
NotModified!	Rows and columns
DataModified!	Rows and columns

When data is retrieved When data is retrieved into a DataWindow, all rows and columns initially have a status of NotModified!.

After data has changed in a column in a particular row, either because the user changed the data or the data was changed programmatically, such as through the SetItem method, the column status for that column changes to DataModified!. Once the status for any column in a retrieved row changes to DataModified!, the row status also changes to DataModified!.

When rows are inserted When a row is inserted into a DataWindow, it initially has a row status of New!, and all columns in that row initially have a column status of NotModified!. After data has changed in a column in the row, either because the user changed the data or the data was changed programmatically, such as through the SetItem method, the column status changes to DataModified!. Once the status for any column in the inserted row changes to DataModified!, the row status changes to NewModified!.

When a DataWindow column has a default value, the column's status does not change to DataModified! until the user makes at least one actual change to a column in that row.

When Update is called A row's status flag determines what SQL command the Update method uses to update the database. INSERT or UPDATE is called, depending upon the following row statuses:

Table 9-6: Effect of row status on SQL command called by Update method

Row status	SQL statement generated
NewModified!	INSERT
DataModified!	UPDATE

A column is included in an UPDATE statement only if the following two conditions are met:

- The column is on the updatable column list maintained by the DataWindow object

  For more information about setting the update characteristics of the DataWindow object, see the *Users Guide.*

- The column has a column status of DataModified!

The DataWindow control includes all columns in INSERT statements it generates. If a column has no value, the DataWindow attempts to insert a NULL. This causes a database error if the database does not allow NULLs in that column.

Changing statuses using SetItemStatus Use SetItemStatus when you want to change the way a row will be updated. Typically, you do this to prevent the default behavior from taking place. For example, you might copy a row from one DataWindow to another. After the user modifies the row, you want to issue an UPDATE statement instead of an INSERT statement.

Changing column status You use SetItemStatus to change the column status from DataModified! to NotModified! or the converse.

Change column status when you change row status
Changing the row status changes the status of all columns in that row to NotModified!, so if the Update method is called, no SQL update is produced. You must change the status of columns to be updated after you change the row status.

Changing row status Changing row status is a little more complicated. The following table illustrates the effect of changing from one row status to another:

Table 9-7: Effect of changing from one row status to another

Original status	***Specified status***			
—	**New!**	**New Modified!**	**Data Modified!**	**Not Modified!**
New!	-	Yes	Yes	No
NewModified!	No	-	Yes	New!
DataModified!	NewModified!	Yes	-	Yes
NotModified!	Yes	Yes	Yes	-

In the table, *Yes* means the change is valid. For example, issuing SetItemStatus on a row that has the status NotModified! to change the status to New! does change the status to New!. *No* means that the change is not valid and the status is not changed.

Issuing SetItemStatus to change a row status from NewModified! to NotModified! actually changes the status to New!. Issuing SetItemStatus to change a row status from DataModified! to New! actually changes the status to NewModified!.

Changing a row's status to NotModified! or New! causes all columns in that row to be assigned a column status of NotModified!. Change the column's status to DataModified! to ensure that an update results in a SQL UPDATE.

Changing the status of a retrieved row from NotModified! to New!
If you change the status of a retrieved row to New! and then make a change to data in a column, *all* the columns in that row change status to DataModified! All the columns change status because the Update method generates a SQL INSERT command that includes the changed data as well as the data that already existed in the other columns.

Changing status indirectly When you cannot change to the desired status directly, you can usually do it indirectly. For example, change New! to DataModified! to NotModified!.

Resetting status for the whole DataWindow object To reset the update status of the entire DataWindow object, use the ResetUpdate method. This sets all status flags to NotModified! except for New! status flags, which remain unchanged.

Examples

This statement sets the status of row 5 in the Salary column of the primary buffer of dw_history to NotModified!:

```
dw_history.SetItemStatus(5, "Salary", &
       Primary!, NotModified!)
```

This statement sets the status of row 5 in the emp_status column of the primary buffer of dw_new_hire to DataModified!:

```
dw_new_hire.SetItemStatus(5, "emp_status", &
       Primary!, DataModified!)
```

This code sets the status of row 5 in the primary buffer of dw_rpt to DataModified! if its status is currently NewModified!:

```
dwItemStatus l_status
l_status = dw_rpt.GetItemStatus(5, 0, Primary!)
IF l_status = NewModified! THEN
       dw_rpt.SetItemStatus(5, 0,
          Primary!, DataModified!)
END IF
```

See also

GetItemStatus
ResetUpdate

SetItemString

Description

Sets the value of a row and column in a DataWindow control to the specified value.

PocketBuilder	✗
PowerBuilder	✓

Syntax

Web DataWindow server component

short *dwcontrol*.**SetItemString** (long *row*, string *column*, string *value*)

short *dwcontrol*.**SetItemStringByColNum** (long *row*, short *column*, string *value*)

Return value

Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is NULL, the method returns NULL.

SetItemTime

Description

Sets the value of a row and column in a DataWindow control to the specified value.

PocketBuilder	✗
PowerBuilder	✓

Syntax

Web DataWindow PSWebDataWindowClass

number *dwcontrol*.**SetItemTime** (number *row*, string *column*, Date *value*)

number *dwcontrol*.**SetItemTime** (number *row*, number *column*, Date *value*)

Web DataWindow server component

short *dwcontrol*.**SetItemTime** (long *row*, string *column*, string *value*)

short *dwcontrol*.**SetItemTimeByColNum** (long *row*, short *column*, string *value*)

Return value

Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is NULL, the method returns NULL.

SetPageSize

Description

Specifies the number of rows to include in a generated Web page for the Web DataWindow.

PocketBuilder	✗
PowerBuilder	✓

Syntax

Web DataWindow server component

string *dwcomponent*.**SetPageSize** (long *pagesize*)

Return value

Returns an empty string if successful and the syntax error message from the Modify method if it fails.

SetPosition

Description

Moves a control within the DataWindow to another band or changes the front-to-back order of controls within a band.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

integer *dwcontrol*.**SetPosition** (string *controlname*, string *band* , boolean *bringtofront*)

Argument	Description
dwcontrol	A reference to a DataWindow control or DataStore.
controlname	The name of the control within the DataWindow that you want to move. You assign names to the controls in the DataWindow painter.
band	A string whose value is the name of the band or layer in which you want to position *controlname*. Layer names are background and foreground. Band names are detail, header, footer, summary, header.#, and trailer.#, where # 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 *controlname* to the front within the band: • TRUE — Bring it to the front. • FALSE — Do not bring it to the front.

Return value

Returns 1 when it succeeds and -1 if an error occurs. If any argument's value is NULL, the method returns NULL.

Usage

For setting the position of controls in the front-to-back order of a PocketBuilder window, see SetPosition in the *PowerScript Reference*.

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)
```

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	✓

Syntax

integer *objectname*.**SetRedraw** (boolean *redraw*)

Argument	Description
objectname	The name of the object or control for which you want to change the redraw setting.
redraw	A boolean value that controls whether PocketBuilder redraws an object automatically after a change. Values are: • TRUE — Automatically redraw the object or control after each change to its properties. • FALSE — Do not redraw after each change.

Return value

Returns 1 if it succeeds and -1 if an error occurs. If *redraw* 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.

Using with other PocketBuilder controls
For use with other objects, see SetRedraw in the *PowerScript Reference*.

SetRow

Description

Sets the current row in a DataWindow control or DataStore.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

integer *dwcontrol*.**SetRow** (long *row*)

Argument	Description
dwcontrol	A reference to a DataWindow control, DataStore, or child DataWindow in which you want to set the current row
row	The row you want to make current

Return value

Returns 1 if it succeeds and -1 if an error occurs. If *row* is less than 1 or greater than the number of rows, SetRow fails. If any argument's value is NULL, the method returns NULL.

Usage

SetRow moves the cursor to the current row but does not scroll the DataWindow control or DataStore.

Events SetRow can trigger these events:

ItemChanged
ItemError
ItemFocusChanged
RowFocusChanged

Avoiding infinite loops
Never call SetRow in the ItemChanged event or any of the other events listed above. Because SetRow can trigger these events, such a recursive call can cause a stack fault.

Examples

This statement sets the current row in dw_employee to 15:

```
dw_employee.SetRow(15)
```

This example unhighlights all highlighted rows, if any. It then sets the current row to 15 and highlights it. If row 15 is not visible, you can use ScrollToRow instead of SetRow:

```
dw_employee.SelectRow(0, FALSE)
dw_employee.SetRow(15)
dw_employee.SelectRow(15, TRUE)
```

See also

GetColumn
GetRow
SetColumn
SetRowFocusIndicator

SetRowFocusIndicator

Description

Specifies the visual indicator that identifies the current row in the DataWindow control. You can use the standard dotted-line rectangle of Windows, PocketBuilder's pointing hand, or an image stored in a PocketBuilder Picture control.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

integer *dwcontrol*.**SetRowFocusIndicator** (RowFocusInd *focusindicator* {, integer *xlocation* {, integer *ylocation* } })

integer *dwcontrol*.**SetRowFocusIndicator** (Picture *picturename* {, integer *xlocation* {, integer *ylocation* } })

Argument	Description
dwcontrol	A reference to a DataWindow control or child DataWindow in which you want to set the row focus indicator.
focusindicator or *picturename*	The visual indicator for the current row. Valid values can be a value of the RowFocusInd enumerated datatype or the name of a Picture control whose image you want to use. For a list of valid enumerated datatype values, see RowFocusInd on page 378.
xlocation (optional)	The x coordinate in PowerBuilder units of the position of the hand or bitmap relative to the upper-left corner of the row.
ylocation (optional)	The y coordinate in PowerBuilder units of the position of the hand or bitmap relative to the upper-left corner of the row.

Return value

Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is NULL, SetRowFocusIndicator returns NULL.

Usage

Sets the current row indicator in *dwcontrol* to *focusindicator*. If you select Hand or a Picture control as the indicator, PocketBuilder displays the indicator at the left side of the body of the DataWindow unless you specify location coordinates (*xlocation*, *ylocation*). The default location is 0,0 (the left side of the body of the DataWindow control).

Pictures as row focus indicators
To use a picture as the row focus indicator, set up the Picture control in the Window painter. Place the Picture control in the window that contains the DataWindow control and then reference it in the SetRowFocusIndicator method. You can hide the picture or place it under the DataWindow control so the user does not see the control itself.

Examples

This statement sets the row focus indicator in dw_employee to the pointing hand:

```
dw_employee.SetRowFocusIndicator(Hand!)
```

If p_arrow is a Picture control in the window, the following statement sets the row focus indicator in dw_employee to p_arrow:

```
dw_employee.SetRowFocusIndicator(p_arrow)
```

See also

GetRow
SetRow

SetSelfLink

Description

Specifies the URL and page parameters for the current page of the Web DataWindow.

PocketBuilder	✗
PowerBuilder	✓

Syntax

Web DataWindow server component

string *dwcomponent*.**SetSelfLink** (string *selflink*, string *selflinkargs*)

Return value

Returns an empty string if successful and the syntax error message from the Modify method if it fails.

SetServerServiceClasses

Description

Tells the server component to trigger custom events defined in user objects for data validation. These user objects, referred to as service classes, must be defined in the PBL or PBD containing the DataWindow object for the server component.

PocketBuilder	✗
PowerBuilder	✓

Syntax

Web DataWindow PSWebDataWindowClass

number *dwcomponent*.**SetServerServiceClasses** (string *serviceclassnames*)

Web DataWindow server component

short *dwcomponent*.**SetServerServiceClasses** (string *serviceclassnames*)

Return value

Returns 1 if it succeeds and -1 if a specified service class does not exist.

SetServerSideState

Description

Tells the server component whether to attempt to maintain its state by saving the retrieved data and leaving the transaction open. Keeping the retrieved data means that the component does not need to reconnect and retrieve data every time a method is called.

PocketBuilder	✗
PowerBuilder	✓

Syntax

Web DataWindow server component

string *dwcomponent*.**SetServerSideState** (boolean *maintainstate*)

Return value

Returns an empty string if it succeeds and an error message from EAServer if it fails.

SetSort

Description

Specifies sort criteria for a DataWindow control or DataStore.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

integer *dwcontrol*.**SetSort** (string *format*)

Argument	Description
dwcontrol	A reference to a DataWindow control, DataStore, or child DataWindow.
format	A string whose value is valid sort criteria for the DataWindow (see Usage). The expression includes column names or numbers. A column number must be preceded by a pound sign (#). If *format* is NULL, PocketBuilder prompts you to enter the sort criteria.

Return value

Returns 1 if it succeeds and -1 if an error occurs.

Usage

A DataWindow object can have sort criteria specified as part of its definition. SetSort overrides the definition, providing new sort criteria for the DataWindow. However, it does not actually sort the rows. Call the Sort method to perform the actual sorting.

The sort criteria for a column has one of the forms shown in the following table, depending on whether you specify the column by name or number. *Order* is either A for ascending or D for descending order. You can specify secondary sorting by specifying criteria for additional columns in the format string. Separate each column specification with a comma.

Table 9-8: Examples for specifying sort order

Syntax for sort order	Examples
columnname order	"emp_lname A" "emp_lname A, dept_id D"
# columnnumber order	"#3 A"

To let the user specify the sort criteria for a DataWindow control, you can pass a null string to the SetSort method. PocketBuilder displays the Specify Sort Columns dialog with the sort specifications blank. Then you can call Sort to apply the user's criteria. You cannot pass a null string to the SetSort method for a DataStore object.

Examples

This statement sets the sort criteria for dw_employee so emp_status is sorted in ascending order and within each employee status, emp_salary is sorted in descending order:

```
dw_employee.SetSort("emp_status A, emp_salary D")
```

If emp_status is column 1 and emp_salary is column 5 in dw_employee, then the following statement is equivalent to the sort specification above:

```
dw_employee.SetSort("#1 A, #5 D")
```

This example defines sort criteria to sort the status column in ascending order and the salary column in descending order within status. After assigning the sort criteria to the DataWindow control dw_emp, it sorts dw_emp:

```
string newsort
newsort = "emp_status A, emp_salary D"
dw_emp.SetSort(newsort)
dw_emp.Sort( )
```

The following example sets the sort criteria for dw_main to null, causing PocketBuilder to display the Specify Sort Columns dialog so that the user can specify sort criteria. The Sort method applies the criteria the user specifies:

```
string null_str
SetNull(null_str)
dw_main.SetSort(null_str)
dw_main.Sort( )
```

See also

Sort

SetSQLPreview

Description

Specifies the SQL statement for a DataWindow control or DataStore that PocketBuilder is about to send to the database.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

integer *dwcontrol*.**SetSQLPreview** (string *sqlsyntax*)

Argument	Description
dwcontrol	A reference to a DataWindow control, DataStore, or child DataWindow.
sqlsyntax	A string whose value is valid SQL syntax for the SQL statement that will be submitted to the database server.

Return value

Returns 1 if it succeeds and 0 if an error occurs. If any argument's value is NULL, the method returns NULL.

Usage

Use SetSQLPreview to modify syntax before you update the database with changes in the DataWindow object.

To obtain the current SQL statement in the SQLPreview event, look at the *sqlsyntax* argument.

When to call SetSQLPreview
Call this method only in the script for the SQLPreview event.

Examples

This statement sets the current SQL string for the DataWindow dw_1:

```
dw_1.SetSQLPreview( &
       "INSERT INTO billings VALUES(100, " + &
       String(Current_balance) + ")")
```

See also

GetSQLPreview
GetUpdateStatus

SetSQLSelect

Description

Specifies the SQL SELECT statement for a DataWindow control or DataStore.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

integer *dwcontrol*.**SetSQLSelect** (string *statement*)

Argument	Description
dwcontrol	The name of the DataWindow control, DataStore, or child DataWindow for which you want to change the SELECT statement.
statement	A string whose value is the SELECT statement for the DataWindow object. The statement must structurally match the current SELECT statement (that is, it must return the same number of columns, the columns must be the same datatype, and the columns must be in the same order).

Return value

SetSQLSelect returns 1 if it succeeds and -1 if the SELECT statement cannot be changed. If any argument's value is NULL, the method returns NULL.

Usage

Use SetSQLSelect to dynamically change the SQL SELECT statement for a DataWindow object in a script.

If the DataWindow is updatable, PocketBuilder validates the SELECT statement against the database and DataWindow column specifications when you call the SetSQLSelect method. Each column in the SQL SELECT statement must match the column type in the DataWindow object. The statement is validated *only* if the DataWindow object is updatable.

You must use the SetTrans or SetTransObject method to set the transaction object before the SetSQLSelect method will execute.

If the new SELECT statement has a different table name in the FROM clause and the DataWindow object is updatable, then PocketBuilder must change the update information for the DataWindow object. PocketBuilder assumes the key columns are in the same positions as in the original definition. The following conditions would make the DataWindow not updatable:

- There is more than one table in the FROM clause
- A DataWindow update column is a computed column in the SELECT statement

If changing the SELECT statement makes the DataWindow object not updatable, the DataWindow control cannot execute an Update method call for the DataWindow object in the future.

Limitations to using SetSQLSelect

Use SetSQLSelect *only* if the data source for the DataWindow object is a SQL SELECT statement *without* retrieval arguments and you want PocketBuilder to modify the update information for the DataWindow object:

```
dw_1.Modify("DataWindow.Table.Select='select...'")
```

Modify does not verify the SELECT statement or change the update information, so it is faster but more susceptible to user error. Although you can use Modify when arguments are involved, this is not recommended because of the lack of verification.

Examples

If the current SELECT statement for dw_emp retrieves no rows, the following statements replace it with the syntax in NewSyn:

```
string OldSyn, NewSyn

OldSyn = &
       'SELECT employee.EMP_Name FROM employee' &
          + 'WHERE salary < 70000'
NewSyn = 'SELECT employee.EMP_Name FROM employee' &
       + 'WHERE salary < 100000'

IF dw_emp.Retrieve( ) = 0 THEN
       dw_emp.SetSQLSelect(NewSyn)
       dw_emp.Retrieve()
END IF
```

See also

Modify
Retrieve
SetTrans
SetTransObject
Update

SetTabOrder

Description

Changes the tab sequence number of a column in a DataWindow control to the specified value.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

integer *dwcontrol*.**SetTabOrder** (integer *column*, integer *tabnumber*)

integer *dwcontrol*.**SetTabOrder** (string *column*, integer *tabnumber*)

Argument	Description
dwcontrol	A reference to a DataWindow control or child DataWindow in which you want to define the tab order.
column	The column to which you are assigning a tab value. *Column* can be a column number or a column name. The column number is the number of the column as it is listed in the Column Specification view of the DataWindow painter—not necessarily the number of the column in the Design view.
tabnumber	The tab sequence number (0 - 9999) you want to assign to the DataWindow column. 0 removes the column from the tab order, which makes it read-only.

Return value

Returns the previous tab value of the column if it succeeds and -1 if an error occurs. If any argument's value is NULL, the method returns NULL.

Usage

You can change a column in a DataWindow object to read-only by changing the tab sequence number of the column to 0.

Examples

This statement changes column 4 of dw_Employee to read-only:

```
dw_Employee.SetTabOrder(4, 0)
```

These statements change column 4 of dw_employee to read-only and later restore the column to its original tab value with read/write status:

```
integer OldTabNum
// Set OldTabNum to the previous tab order value
OldTabNum = dw_employee.SetTabOrder(4, 0)
... // Some processing
// Return column 4 to its previous tab value.
dw_employee.SetTabOrder(4, OldTabNum)
```

SetText

Description

Replaces the text in the edit control over the current row and column in a DataWindow control or DataStore.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

integer *dwcontrol*.**SetText** (string *text*)

Argument	Description
dwcontrol	The name of the DataWindow control or DataStore in which you want to specify the text in the current row and column.
text	A string whose value you want to put in the current row and column. The value must be compatible with the datatype of the column.

Return value

Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is NULL, the method returns NULL.

Usage

SetText only sets the value in the edit control. When the user changes focus to another row and column, PocketBuilder accepts the text as the item in the row and column.

In the ItemChanged or ItemError event, PocketBuilder or your own script might determine that the value in the edit control is invalid or needs further processing. You can call SetItem to specify a new item value for the row and column. After calling SetItem, you can call SetText to put that same value in the edit control so that the user also sees the value. In the script, use a return code that rejects the value in the edit control, avoiding further processing, and allow the focus to change. (Return 2 for ItemChanged and 3 for ItemError.)

Examples

These statements replace the value of the current row and column in dw_employee with Tex and then call AcceptText to accept and move Tex into the current column. (Do not use this code in the ItemChanged or ItemError event because it calls AcceptText.)

```
dw_employee.SetText("Tex")
dw_employee.AcceptText()
```

This example converts a number that the user enters in the column called credit to a negative value and sets both the item and the edit control's text to the negative number. This code is the script for the ItemChanged event. The data argument holds the newly entered value:

```
integer negative

IF dwo.Name = "credit" THEN
       IF Integer(data) > 0 THEN
          // Convert to negative if it's positive
          negative = Integer(data) * -1

          // Change the primary buffer value.
          This.SetItem(row, "credit", negative)

          // Change the value in the edit control
          This.SetText(String(negative))
          RETURN 2
       END IF
END IF
```

See also

AcceptText
GetText

SetTrans

Specifies connection information for a DataWindow or DataStore.

To specify connection information	Use
Using values from an external transaction object	Syntax 1
For the Web DataWindow server component	Syntax 2

Syntax 1 Using values from an external transaction object

Description

Sets the values in the internal transaction object for a DataWindow control or DataStore to the values from the specified transaction object. The transaction object supplies connection settings, such as the database name.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

integer *dwcontrol*.**SetTrans** (transaction *transaction*)

Argument	Description
dwcontrol	A reference to a DataWindow control, DataStore, or child DataWindow in which you want to set the values of the internal transaction object
transaction	The name of the transaction object from which you want *dwcontrol* to get values

Return value

Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is NULL, the method returns NULL.

Usage

In most cases, use the SetTransObject method to specify the transaction object. It is more efficient and allows you to control when changes get committed to the database.

SetTrans copies the values from a specified transaction object to the internal transaction object for the DataWindow control or DataStore. When you use SetTrans in a script, the DataWindow uses its internal transaction object and automatically connects and disconnects as needed; any errors that occur cause an automatic rollback. With SetTrans, you do not specify SQL statements, such as CONNECT, COMMIT, and DISCONNECT. The DataWindow control connects and disconnects after each Retrieve or Update function.

Use SetTrans when you want PocketBuilder to manage the database connections automatically because you have a limited number of available connections or expect to use the application from a remote location. SetTrans is appropriate when you are only retrieving data and do not need to hold database locks on records the user is modifying. For better performance, however, you should use SetTransObject.

DBMS connection settings You must set the parameters required to connect to your DBMS in the transaction object before you can use the transaction object to set the DataWindow's internal transaction object and connect to the database.

Updating more than one table When you use SetTrans to specify the transaction object, you cannot update multiple DataWindow objects or multiple tables within one object.

Examples

This statement sets the values in the internal transaction object for dw_employee to the values in the default transaction object SQLCA:

```
dw_employee.SetTrans(SQLCA)
```

The following statements change the database type and password of dw_employee. The first two statements create the transaction object emp_TransObj. The next statement uses the GetTrans method to store the values of the internal transaction object for dw_employee in emp_TransObj. The next two statements change the database type and password. The SetTrans method assigns the revised values to dw_employee:

```
// Name the transaction object.
transaction emp_TransObj

// Create the transaction object.
emp_TransObj = CREATE transaction

// Fill the new object with the original values.
dw_employee.GetTrans(emp_TransObj)
// Change the database type.
emp_TransObj.DBMS ="Sybase"
// Change the password.
emp_TransObj.LogPass = "cam2"

// Put the revised values into the
// DataWindow transaction object.
dw_employee.SetTrans(emp_TransObj)
```

See also

GetTrans
SetTransObject

Syntax 2 For the Web DataWindow server component

Description

Specifies connection information for the Web DataWindow, such as the database name.

PocketBuilder	✗
PowerBuilder	✓

Syntax

Web DataWindow server component

integer *dwcontrol*.**SetTrans** (string *dbms*, string *dbparm*, string *lock*, string *logid*, string *logpass*, string *database*, string *servername*)

Return value

Returns 1 if it succeeds and -1 if an error occurs.

SetTransObject

Description

Causes a DataWindow control or DataStore to use a programmer-specified transaction object. The transaction object provides the information necessary for communicating with the database.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

integer *dwcontrol*.**SetTransObject** (transaction *transaction*)

Argument	Description
dwcontrol	A reference to a DataWindow control, DataStore, or child DataWindow in which you want to use a programmer-specified transaction object rather than the DataWindow control's internal transaction object
transaction	The name of the transaction object you want to use in the *dwcontrol*

Return value

Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is NULL, the method returns NULL.

Usage

Transaction objects in PocketBuilder A programmer-specified transaction object gives you more control over the database transactions and provides efficient application performance. You control the database connection by using SQL statements such as CONNECT, COMMIT, and ROLLBACK.

Since the DataWindow control does not have to connect to the database for every RETRIEVE and UPDATE statement, these statements run faster. You are responsible for committing and rolling back transactions after you call the Update method, using code like the following:

```
IF dw_Employee.Update()>0 THEN
       COMMIT USING emp_transobject;
ELSE
       ROLLBACK USING emp_transobject;
END IF
```

You must set the parameters required to connect to your DBMS in the transaction object before you can use the transaction object to connect to the database. PocketBuilder provides a global transaction object called SQLCA, which is all you need if you are connecting to one database. You can also create additional transaction objects, as shown in the examples.

To use SetTransObject, write code that does the following tasks:

1 Set up the transaction object by assigning values to its fields (usually in the application's Open event).

2 Connect to the database using the SQL CONNECT statement and the transaction object (in the Open event for the application or window).

3 Call SetTransObject to associate the transaction object with the DataWindow control or DataStore (usually in the window's Open event).

4 Check the return value from the Update method and follow it with a SQL COMMIT or ROLLBACK statement, as appropriate.

If you change the DataWindow object associated with the DataWindow control (or DataStore), or if you disconnect and reconnect to a database, the connection between the DataWindow control (or DataStore) and the transaction object is severed. You must call SetTransObject again to reestablish the connect.

SetTransObject versus SetTrans

In most cases, use the SetTransObject method to specify the transaction object because it is efficient and gives you control over when transactions are committed.

The SetTrans method provides another way of managing the database connection. SetTrans, which sets transaction information in the internal transaction object for the DataWindow control or DataStore, manages the connection automatically. You do not explicitly connect to the database; the DataWindow connects and disconnects for each database transaction, which is less efficient but necessary in some situations.

For more information, see SetTrans.

Examples

This statement causes dw_employee to use the default transaction object SQLCA:

```
dw_employee.SetTransObject(SQLCA)
```

This statement causes dw_employee to use the programmer-defined transaction object emp_TransObj. In this example, emp_TransObj is an instance variable, but your script must allocate memory for it with the CREATE statement before you use it:

```
emp_TransObj = CREATE transaction
... // Assign values to the transaction object
dw_employee.SetTransObject(emp_TransObj)
```

See also

GetTrans
SetTrans

SetValidate

Description

Sets the input validation rule for a column in a DataWindow control or DataStore.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

integer *dwcontrol*.**SetValidate** (string *column*, string *rule*)

integer *dwcontrol*.**SetValidate** (integer *column*, string *rule*)

Argument	Description
dwcontrol	A reference to a DataWindow control, DataStore, or child DataWindow.
column	The column for which you want to set the input validation rule. *Column* can be a column number or a column name.
rule	A string whose value is the validation rule for validating the data.

Return value

Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is NULL, the method returns NULL.

Usage

Validation rules are boolean expressions that usually compare the value in the column's edit control to some other value. When data the user enters fails to meet the criteria established in the validation rule, an ItemError event occurs.

You can specify validation rules in the Database painter or the DataWindow painter, and you can change the rules in scripts using SetValidate. A validation rule can include any DataWindow painter function.

For more information, see the *Users Guide*.

If you want to change a column's validation rule temporarily, you can use GetValidate to get and save the current rule. To include the value the user entered in the validation rule, use the GetText method. You can compare its return value to the validation criteria.

If the validation rule contains numbers, the DataWindow expects the numbers in U.S. format. In PocketBuilder, be aware that the String function formats numbers using the current system settings. If you use it to build the rule, specify a display format that produces U.S. notation.

Examples

The following assigns a validation rule to the current column in dw_employee. The rule ensures that the data entered is greater than zero:

```
dw_employee.SetValidate(dw_employee.GetColumn(), &
```

```
        "Number(GetText( )) > 0")
```

The following assigns a validation rule to the current column in dw_employee. The rule checks that the value entered is less than the value in the Full_Price column:

```
dw_employee.SetValidate(dw_employee.GetColumn(), &
        "Number(GetText( )) < Full_Price")
```

This example defines a new validation rule for the column emp_state in the DataWindow control dw_employee. The new rule is [A-Z]+ , meaning the data in emp_state must be all uppercase characters. The text pattern must be enclosed in quotes within the quoted validation rule. The embedded quotes are specified with ~". The script saves the old rule, assigns the new rule, performs some processing, and then sets the validation rule back to the old rule:

```
string OldRule, NewRule

NewRule = "Match(GetText(), ~"[A-Z]+~")"

OldRule = dw_employee.GetValidate("emp_state")

dw_employee.SetValidate("emp_state", NewRule)
... //Process data using the new rule.

// Set the validation rule back to the old rule.
dw_employee.SetValidate("emp_state", OldRule)
```

See also GetValidate

SetValue

Description Sets the value of an item in a value list or code table for a column in a DataWindow control or DataStore. (A value list is called a code table when it has both display and data values.) SetValue does not affect the data stored in the column.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax integer *dwcontrol*.**SetValue** (string *column*, integer *index*, string *value*)

integer *dwcontrol*.**SetValue** (integer *column*, integer *index*, string *value*)

Argument	Description
dwcontrol	A reference to a DataWindow control or DataStore.
column	The column that contains the value list or code table. *Column* can be a column number or a column name. The edit style of the column can be DropDownListBox, Edit, or RadioButton. SetValue has no effect when *column* has the EditMask or DropDownDataWindow edit style.
index	The number of the item in the value list or code table for which you want to set the value.
value	A string whose value is the new value for the item. For a code table, use a tab (~t in PocketBuilder) to separate the display value from the data value ("Texas~tTX"). The data value must be a string that can be converted to the datatype of the column.

Return value

Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is NULL, the method returns NULL.

Examples

This statement sets the value of item 3 in the value list for the column emp_state of dw_employee to Texas:

```
dw_employee.SetValue("emp_state", 3, "Texas")
```

This statement sets the display value of item 3 in the code table for the column named emp_state of dw_employee to Texas and the data value to TX:

```
dw_employee.SetValue("emp_state", 3, "Texas~tTX")
```

The following statements use a SQL cursor and FETCH statement to populate the ListBox portion of a DropDownListBox style column called product_col of a DataWindow object with code table values:

```
integer prod_code, i = 1
string prod_name

DECLARE prodcur CURSOR FOR
       SELECT product.name, product.code
       FROM product USING SQLCA;

CONNECT USING SQLCA;
IF SQLCA.SQLCode <> 0 THEN
       MessageBox("Status","Connect Failed " &
          + SQLCA.SQLErrText)
       RETURN
END IF

OPEN prodcur;
```

```
IF SQLCA.SQLCode <> 0 THEN
       MessageBox("Status","Cursor Open Failed " &
           + SQLCA.SQLErrText)
       RETURN
END IF

FETCH prodcur INTO :prod_name, :prod_code;

DO WHILE SQLCA.SQLCode = 0
       dw_products.SetValue("product_col", i, &
           prod_name + "~t" + String(prod_code))
       i = i + 1
       FETCH prodcur INTO :prod_name, :prod_code;
LOOP

CLOSE prodcur;
DISCONNECT USING SQLCA;
```

See also GetValue

SetWeight

Description Specifies the types of JavaScript code that will be included in the generated HTML.

PocketBuilder	✗
PowerBuilder	✓

Syntax

Web DataWindow PSWebDataWindowClass

number *dwcomponent*.**SetWeight** (boolean *allowupdate*, boolean *validation*, boolean *events*, boolean *clientscriptable*, boolean *clientformatting*)

Web DataWindow server component

integer *dwcomponent*.**SetWeight** (boolean *allowupdate*, boolean *validation*, boolean *events*, boolean *clientscriptable*, boolean *clientformatting*)

Return value Returns an empty string if successful and the syntax error message from the Modify method if it fails.

ShareData

Description

Shares data retrieved by one DataWindow control (or DataStore), which is referred to as the primary DataWindow, with another DataWindow control (or DataStore), referred to as the secondary DataWindow.

The controls do not share formatting; only the data is shared, including data in the primary buffer, the delete buffer, the filter buffer, and the sort order.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

integer *dwprimary*.**ShareData** (datawindow *dwsecondary*)

integer *dwprimary*.**ShareData** (datastore *dwsecondary*)

integer *dwprimary*.**ShareData** (datawindowchild *dwsecondary*)

Argument	Description
dwprimary	The name of the primary DataWindow. The primary DataWindow is the owner of the data. When you destroy this DataWindow, the data disappears. *Dwprimary* can be a child DataWindow.
dwsecondary	The name of the secondary DataWindow with which the control *dwprimary* will share the data. The secondary DataWindow cannot be a Crosstab DataWindow. It can be a child DataWindow.

Return value

Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is NULL, ShareData returns NULL.

Usage

The columns must be the same for the DataWindow objects in the primary and secondary DataWindow controls, but the SELECT statements may be different. For example, you could share data between DataWindow objects with these SELECT statements:

```
SELECT dept_id from dept

SELECT dept_id from dept where dept_id = 200

SELECT dept_id from employee
```

WHERE clause in secondary has no effect
The WHERE clause in the DataWindow object in the secondary DataWindow control has no effect on the number of rows returned. The number of rows returned to both DataWindow controls is determined by the WHERE clause in the primary DataWindow object.

You could also share data with a DataWindow object that has an external data source and columns defined to be like the columns in the primary. To share data between a primary DataWindow and more than one secondary DataWindow control, call ShareData for each secondary DataWindow control.

ShareData shares only the primary buffer of the primary DataWindow with the primary buffer of the secondary DataWindow. A DropDownDataWindow in the secondary DataWindow will not display any data unless you explicitly populate it. You can do this by getting a handle to the DropDownDataWindow (by calling the GetChild method) and either retrieving the DropDownDataWindow or using ShareData to share data from an appropriate data source with the DropDownDataWindow.

To turn off sharing in a primary or secondary DataWindow, call the ShareDataOff method. When sharing is turned off for the primary DataWindow, the secondary DataWindows are disconnected and the data disappears. However, turning off sharing for a secondary DataWindow does not affect the data in the primary DataWindow or other secondary DataWindows.

When you call methods in either the primary or secondary DataWindow that change the data, PocketBuilder applies them to the primary DataWindow control and all secondary DataWindow controls are affected.

For example, when you call any of the following methods for a secondary DataWindow control, PocketBuilder applies it to the primary DataWindow. Therefore, all messages normally associated with the method go to the primary DataWindow control. Such methods include:

DeleteRow
Filter
GetSQLSelect
ImportFile
ImportString
ImportClipboard
InsertRow
ReselectRow
Reset

Retrieve
SetFilter
SetSort
SetSQLSelect
Sort
Update

Computed fields in secondary DataWindow controls
A secondary DataWindow control can have only data that is in the primary DataWindow control. If you add a computed field to a secondary control, it will not display when you run the application unless you also add it to the primary control.

Query mode and secondary DataWindows
When you are sharing data, you cannot turn on query mode for a secondary DataWindow. Trying to set the QueryMode or QuerySort DataWindow object properties results in an error.

Examples

In this example, the programmer wants to allow the user to view two portions of the same data retrieved from the database and uses the ShareData method to accomplish this in the script for the Open event for the window. The SELECT statement for both DataWindow objects is the same, but the DataWindow object in dw_dept displays only two of the five columns displayed in dw_employee:

```
CONNECT USING SQLCA;
dw_employee.SetTransObject(SQLCA)
dw_employee.Retrieve()
dw_employee.ShareData(dw_dept)
```

See also

ShareDataOff

ShareDataOff

Description

Turns off the sharing of data buffers for a DataWindow control or DataStore.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

integer *dwcontrol*.**ShareDataOff** ( )

Argument	Description
dwcontrol	A reference to a DataWindow control, DataStore, or child DataWindow

Return value

Returns 1 if it succeeds and -1 if an error occurs. If *dwcontrol* is NULL, ShareDataOff returns NULL.

Usage

Two or more DataWindow controls (or DataStores) can share data. See ShareData for more information about shared data buffers and primary and secondary DataWindows.

When you call ShareDataOff for a secondary DataWindow, that control no longer contains data, but the primary DataWindow and other secondary controls are not affected. When you call ShareDataOff for the primary DataWindow, all secondary DataWindows are disconnected and no longer contain data.

Examples

These statements establish the sharing of data among three DataWindow controls and then turn off sharing for one of the secondary DataWindow controls:

```
CONNECT USING SQLCA;
dw_corp.SetTransObject(SQLCA)
dw_corp.Retrieve()
dw_corp.ShareData(dw_emp)
dw_corp.ShareData(dw_dept)
... // Some processing
dw_emp.ShareDataOff()
```

See also

ShareData

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	✓

Syntax

integer *objectname*.**Show** ( )

Argument	Description
objectname	The name of the object or control you want to make visible (show)

Return value

Returns 1 if it succeeds and -1 if an error occurs. If *objectname* is NULL, Show returns NULL.

Usage

Inherited from GraphicObject. For details on use with other PocketBuilder objects, see Show in the *PowerScript Reference*.

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	✓

Syntax

PowerBuilder DataWindow control

integer *rtename*.**ShowHeadFoot** (boolean *editheadfoot*)

Return value

Returns 1 if it succeeds and -1 if an error occurs.

Sort

Description

Sorts the rows in a DataWindow control or DataStore using the DataWindow's current sort criteria.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

integer *dwcontrol*.**Sort** ( )

Argument	Description
dwcontrol	A reference to a DataWindow control, DataStore, or child DataWindow

Return value

Returns 1 if it succeeds and -1 if an error occurs. If *dwcontrol* is NULL, Sort returns NULL.

Usage

Sort uses the current sort criteria for the DataWindow. To change the sort criteria, use the SetSort method. The SetSort method is equivalent to using the Sort command on the Rows menu of the DataWindow painter. If you do not call SetSort to set the sort criteria before you call Sort, Sort uses the sort criteria specified in the DataWindow object definition.

When the Retrieve method retrieves data for the DataWindow, PocketBuilder applies the sort criteria that were defined for the DataWindow object, if any. You need to call Sort only after you change the sort criteria with SetSort or if the data has changed because of processing or user input.

For information on letting the user specify sort criteria using the built-in dialog box, see SetSort on page 624.

When you sort a DataWindow on a specified column, rows with NULL data remain at the top, regardless of whether you choose ascending or descending order for your sort criteria. The sort order is performed on a result set returned from a database, but is not necessarily the same sort order used by the database (to return the result set) when an ORDER BY clause is used in a SQL query. The Sort method uses a typical lexical sort, with symbols, such as a hyphen or underline, ranked higher than alphanumeric characters.

When the Retrieve As Needed option is set, the Sort method cancels its effect. Sort causes all rows to be retrieved so that they are sorted correctly. It also changes the current row to 1 without causing the RowFocusChanged or RowFocusChanging events to fire. These events should be triggered programmatically after the Sort function is called.

Sort has no effect on the DataWindows in a composite report.

Sorting and groups
To sort a DataWindow object with groups, call GroupCalc after you call Sort.

Using with other controls
For use with PocketBuilder ListView and TreeView controls, see Sort in the *PowerScript Reference*.

Examples

This example sets dw_employee to be sorted by column 1 ascending and then by column 2 descending. Then it sorts the rows:

```
dw_employee.SetRedraw(false)
dw_employee.SetSort("#1 A, #2 D")
dw_employee.Sort()
dw_employee.SetRedraw(true)
```

In this example, the rows in the DataWindow dw_depts are grouped based on department and the rows are sorted based on employee name. If the user has changed the department of several employees, then the following commands apply the sort criteria so that each group is in alphabetical order and then regroup the rows:

```
dw_depts.SetRedraw(false)
dw_depts.Sort()
dw_depts.GroupCalc()
dw_depts.SetRedraw(true)
```

See also

GroupCalc
SetSort

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	✓

Syntax

string *editname*.**TextLine** ( )

Argument	Description
editname	A reference to a DataWindow control

Return value

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

TextLine reports information about the edit control over the current row and column.

Using with other controls
For use with other PocketBuilder controls, see TextLine in the *PowerScript Reference*.

Examples

In the DataWindow control dw_letter, if the insertion point is on line 4 in the edit control and the text on the line is North Carolina, then this example sets linetext to North Carolina:

```
string linetext
linetext = dw_letter.TextLine()
```

See also

SelectTextLine

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	✓

Syntax

integer *objectname*.**TriggerEvent** (trigevent *event* {, long *word*, long *long* })

integer *objectname*.**TriggerEvent** (trigevent *event* {, long *word*, string *long* })

Argument	Description
objectname	The name of any PocketBuilder object or control that has events associated with it.

Argument	Description
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 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 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

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

Inherited from PowerObject. For information, see TriggerEvent in the *PowerScript Reference*.

See also

Post in the *PowerScript Reference*
PostEvent in the *PowerScript Reference*
Send in the *PowerScript Reference*

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	✓

Syntax

object *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

Inherited from PowerObject. For information, see TypeOf in the *PowerScript Reference*.

See also

ClassName

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	✓

Syntax

integer *editname*.**Undo** ( )

Argument	Description
editname	A reference to a DataWindow control. Reverses the last edit in the edit control over the current row and column.

Return value

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 method.

Using with other controls
For examples and for use with other PocketBuilder controls, see Undo in the *PowerScript Reference*.

See also

CanUndo

Update

Description

Updates the database with the changes made in a DataWindow control or DataStore. Update can also call AcceptText for the current row and column before it updates the database.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

integer *dwcontrol*.**Update** ({ boolean *accept* {, boolean *resetflag* } })

Argument	Description
dwcontrol	A reference to a DataWindow control, DataStore, or child DataWindow.
accept (optional)	A boolean value specifying whether the DataWindow control or DataStore should automatically perform an AcceptText prior to performing the update: • TRUE — (Default) Perform AcceptText. The update is canceled if the data fails validation. • FALSE — Do not perform AcceptText.
resetflag (optional)	A boolean value specifying whether *dwcontrol* should automatically reset the update flags: • TRUE — (Default) Reset the flags. • FALSE — Do not reset the flags.

Return value

Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is NULL, Update returns NULL. If there is no DataWindow object assigned to the DataWindow control or DataStore, this method returns 1.

Usage

You must use the SetTrans or the SetTransObject method to specify the database connection before the Update method will execute. When you use SetTransObject, the more efficient of the two, you must do your own transaction management, which includes issuing the SQL COMMIT or ROLLBACK statement to finalize the update.

Test success/failure code

It is good practice to test the success/failure code after calling Update. You can also verify the number of rows inserted, updated, and deleted by a DataWindow update by examining the values of the arguments of the UpdateEnd event.

By default, Update resets the update flags after successfully completing the update. However, you can prevent the flags from being reset until you perform other validations and commit the changes. When you are satisfied with the update, call ResetUpdate to clear the flags so that items are no longer marked as modified.

Use SetTransObject when *resetflag* is FALSE
You would typically use SetTransObject, not SetTrans, to specify the transaction object for the DataWindow control or DataStore when you plan to update with the *resetflag* argument set to FALSE. Only SetTransObject allows you to control when changes are committed.

If you want to update several tables in one DataWindow control or DataStore, you can use Modify to change the Update property of columns in each table. To preserve the status flags of the rows and columns, set the *resetflag* argument to FALSE. Because the updates all occur in the same DataWindow control or DataStore, you cannot allow the flags to be cleared until all the tables have used them. When all the updates are successfully completed and committed, you can call ResetUpdate to clear the changed flags in the DataWindow. For an example of this technique, see Modify.

If you are updating multiple DataWindow controls or DataStores as part of one transaction, set the *resetflag* argument to FALSE. This will prevent the DataWindow from "forgetting" which rows to update in case one of the updates fails. You can roll back, try to correct the situation, and update again. Once all of the DataWindows have been updated successfully, use COMMIT to finalize the transaction and use ResetUpdate to reset the DataWindow's status flags.

If you call Update with the *resetflag* argument set to FALSE and do not call ResetUpdate, the DataWindow will attempt to issue the same SQL statements again the next time you call Update.

Caution
If you call Update in an ItemChanged event, be sure to set the accept argument to FALSE to avoid an endless loop and a stack fault. Because AcceptText triggers an ItemChanged event, you cannot call it in that event (see AcceptText on page 434).

If you call Update in the ItemChanged event, then the item's old value is updated in the database, not the newly entered value. The newly entered value in the edit control is still being validated and does not become the item value until the ItemChanged event is successfully completed. If you want to include the new value in an update in the ItemChanged event, use the appropriate SetItem method first.

Apply GetChanges after deleting rows in a distributed application
If a DataWindow or data store is populated using SetChanges or SetFullState, and an Update is done that includes deleted rows, the deleted rows remain in the delete buffer until a subsequent GetChanges is applied to the DataWindow or data store.

Events Update can trigger these events:

DBError
SQLPreview
UpdateEnd
UpdateStart

If AcceptText is performed, it can trigger these events:

ItemChanged
ItemError

Examples

This example connects to the database, specifies a transaction object for the DataWindow control with SetTransObject, and then updates the database with the changes made in dw_employee. By default, AcceptText is performed on the data in the edit control for the current row and column and the status flags are reset:

```
CONNECT USING SQLCA;
dw_employee.SetTransObject(SQLCA)
... // Some processing
dw_employee.Update()
```

This example connects to the database, specifies a transaction object for the DataWindow control with SetTransObject, and then updates the database with the changes made in dw_employee. The update resets the status flags but does not perform AcceptText before updating the database:

```
CONNECT USING SQLCA;
dw_employee.SetTransObject(SQLCA)
... // Some processing
dw_Employee.Update(FALSE, TRUE)
```

As before, this example connects to the database, specifies a transaction object for the DataWindow control with SetTransObject, and then updates the database with the changes made in dw_employee. After Update is executed, the example checks the return code and, depending on the success of the update, executes a COMMIT or ROLLBACK:

```
integer rtn

CONNECT USING SQLCA;
dw_employee.SetTransObject(SQLCA)
rtn = dw_employee.Update()

IF rtn = 1 THEN
       COMMIT USING SQLCA;
ELSE
       ROLLBACK USING SQLCA;
END IF
```

See also

AcceptText
Modify
ResetUpdate
Print
SaveAs
SetTrans
SetTransObject

CHAPTER 10

Methods for Graphs in the DataWindow Control

About this chapter

This chapter documents the methods that you can use to manipulate DataWindow graphs and provides syntax, notes, and examples for these methods.

Other methods for DataWindows and DataStores are in a separate chapter.

Contents

The graph methods are in alphabetical order.

CategoryCount

Description

Counts the number of categories on the category axis of a graph.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

integer *dwcontrol*.**CategoryCount** (string *graphcontrol*)

Argument	Description
dwcontrol	A reference to a DataWindow control containing the graph
graphcontrol	A string whose value is the name of the graph in the DataWindow for which you want the number of categories

Return value

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")
```

See also

DataCount
SeriesCount

CategoryName

Description

Obtains the category name associated with the specified category number.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

string *dwcontrol*.**CategoryName** (string *graphcontrol*, integer *categorynumber*)

Argument	Description
dwcontrol	A reference to the DataWindow control containing the graph
graphcontrol	A string whose value is the name of the graph in the DataWindow for which you want the name of a specific category
categorynumber	The number of the category for which you want the name

Return value

Returns the name of *categorynumber* in the graph named in *graphcontrol*. 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_revenues in the DataWindow control dw_findata:

```
string ls_name
ls_name = &
       dw_findata.CategoryName("gr_revenues", 5)
```

See also

CategoryCount
SeriesName

Clipboard

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	✓

Syntax

integer *dwcontrol*.**Clipboard** (string *graphcontrol*)

Argument	Description
dwcontrol	A reference to the DataWindow control containing the graph
graphcontrol	A string whose value is the name of the graph in the DataWindow object

Return value

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_employees in the DataWindow control dw_emp_data to the clipboard:

```
dw_emp_data.Clipboard("gr_employees")
```

See also

Clipboard in the *PowerScript Reference*
Copy

DataCount

Description

Reports the number of data points in the specified series in a graph.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

long *dwcontrol*.**DataCount** (string *graphcontrol*, string *seriesname*)

Argument	Description
dwcontrol	A reference to the DataWindow control containing the graph
graphcontrol	The name of the graph in the DataWindow control
seriesname	A string whose value is the name of the series for which you want the number of data points

Return value

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 Salary in the graph gr_dept in the DataWindow control dw_employees:

```
long ll_count
ll_count = &
       dw_employees.DataCount("gr_dept", "Salary")
```

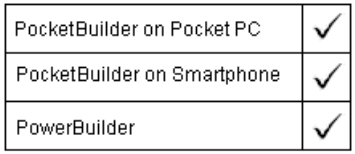
See also

SeriesCount

FindCategory

Description

Obtains the number of a category in a graph when you know the category's label. The category values label the category axis.



Syntax

integer *dwcontrol*.**FindCategory** (string *graphcontrol*, date *categoryvalue*)

integer *dwcontrol*.**FindCategory** (string *graphcontrol*, datetime *categoryvalue*)

integer *dwcontrol*.**FindCategory** (string *graphcontrol*, double *categoryvalue*)

integer *dwcontrol*.**FindCategory** (string *graphcontrol*, string *categoryvalue*)

integer *dwcontrol*.**FindCategory** (string *graphcontrol*, time *categoryvalue*)

Argument	Description
dwcontrol	A reference to the DataWindow control containing the graph.
graphcontrol	A string whose value is the name of the graph in the DataWindow control.
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

Returns the number of the category named in *categoryvalue* in the graph. 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 the category named Qty in the graph gr_computers in the DataWindow control dw_equipment:

```
integer CategoryNbr
CategoryNbr = &
       dw_equipment.FindCategory("gr_computers", "Qty")
```

See also

FindSeries

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	✓

Syntax

integer *dwcontrol*.**FindSeries** (string *graphcontrol*, string *seriesname*)

Argument	Description
dwcontrol	A reference to the DataWindow control containing the graph
graphcontrol	A string whose value is the name of the graph in the DataWindow control
seriesname	A string whose value is the name of the series for which you want the number

Return value

Returns the number of the series named in *seriesname* in the graph. 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. Use FindSeries when you know only a series' name or when the numbering may have changed.

Examples

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

FindCategory

GetData

Description

Gets the value of a data point in a series in a graph when the values axis has numeric values.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

double *dwcontrol*.**GetData** (string *graphcontrol*, integer *seriesnumber*, long *datapoint* , { grDataType *datatype* })

Argument	Description
dwcontrol	A reference to the DataWindow control containing the graph.
graphcontrol	A string whose value is the name of the graph in the DataWindow control.
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 (optional argument, for scatter graph only)	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. For more information, see grDataType on page 375.

Return value

Returns the value of the data in *datapoint* if it succeeds, 0 if the series does not exist, and -1 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 method 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 x value of the data point in the scatter graph gr_sales_yr in the DataWindow dw_sales and store it in data_value:

```
integer SeriesNbr, ItemNbr
double data_value

dw_sales.ObjectAtPointer("gr_sales_yr", SeriesNbr, &
       ItemNbr)
data_value = dw_sales.GetData("gr_sales_yr", &
       SeriesNbr, ItemNbr, xValue!)
```

See also

FindSeries
GetDataValue
ObjectAtPointer

GetDataDateVariable

Description

Returns the value associated with a data point in a graph in a DataWindow object when the values axis has the date datatype. You must call GetDataDate first to retrieve the line style information. (GetDataDate is based on GetDataValue and is documented in that entry.)

PocketBuilder	✗
PowerBuilder	✓

Syntax

Web ActiveX

Date *dwcontrol*.**GetDataDateVariable** ( )

Return value

Returns a date value associated with a data point in a graph.

GetDataNumberVariable

Description

Returns the value associated with a data point in a graph in a DataWindow object when the values axis has a numeric datatype. You must call GetDataNumber first to retrieve the line style information. (GetDataNumber is based on GetDataValue and is documented in that entry.)

PocketBuilder	✗
PowerBuilder	✓

Syntax

Web ActiveX

number *dwcontrol*.**GetDataNumberVariable** ( )

Return value

Returns a number value associated with a data point in a graph.

GetDataPieExplode

Description

Reports the percentage of the pie graph's radius that a pie slice is moved away from the center of the pie graph. 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	✓

Syntax

integer *dwcontrol*.**GetDataPieExplode** (string *graphcontrol*, integer *series*, integer *datapoint*, REF integer *percentage*)

Argument	Description
dwcontrol	A reference to the DataWindow control containing the graph
graphcontrol	A string whose value is the name of the graph in the DataWindow control
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 that the pie slice is exploded

Return value

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 GraphType. 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 control:

```
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("gr_sales_yr", series, &
           datapoint, percentage)
       MessageBox("Explosion Percentage", &
           "Data point " + This.CategoryName(datapoint)&
           + " in series " + This.SeriesName(series) &
           + " is exploded " + String(percentage) + "%")
END IF
```

See also

GetDataPieExplodePercentage
SetDataPieExplode

GetDataPieExplodePercentage

Description

Returns the percentage value that a slice is exploded in a pie graph in a DataWindow object. You must call GetDataPieExplode first to retrieve the information and then call this method to get the value.

PocketBuilder	✗
PowerBuilder	✓

Syntax

Web ActiveX.

number *dwcontrol*.**GetDataPieExplodePercentage** ( )

Return value

Returns a number specifying how much the pie slice is exploded.

GetDataStringVariable

Description

Returns the value associated with a data point in a graph in a DataWindow object when the values axis has the string datatype. You must call GetDataString first to retrieve the line style information. (GetDataString is based on GetDataValue and is documented in that entry.)

PocketBuilder	✗
PowerBuilder	✓

Syntax

Web ActiveX

string *dwcontrol*.**GetDataStringVariable** ( )

Return value

String. Returns a string value associated with a data point in a graph.

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 (called GetDataStyleColor in JavaScript)	Syntax 1
Line style and width used by the data point (called GetDataStyleLine in JavaScript)	Syntax 2
Fill pattern for the data point (called GetDataStyleFill in JavaScript)	Syntax 3
Symbol for the data point (called GetDataStyleSymbol in JavaScript)	Syntax 4

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.

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	✓

Syntax

integer *dwcontrol*.**GetDataStyle** (string *graphcontrol*, integer *seriesnumber*, integer *datapointnumber*, grColorType *colortype*, REF long *colorvariable*)

Argument	Description
dwcontrol	A reference to the DataWindow control containing the graph.
graphcontrol	A string whose value is the name of the graph in the DataWindow control.
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. For a list of values, see grColorType on page 374.
colorvariable	A long variable in which you want to store the color.

Return value

Returns 1 if it succeeds and -1 if an error occurs. GetDataStyle stores an RGB color value in *colorvariable*. If any argument's value is NULL, GetDataStyle returns NULL.

Examples

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	✓

Syntax

integer *dwcontrol*.**GetDataStyle** (string *graphcontrol*, integer *seriesnumber*, integer *datapointnumber*, REF LineStyle *linestyle*, REF integer *linewidth*)

Argument	Description
dwcontrol	A reference to the DataWindow control containing the graph.
graphcontrol	A string whose value is the name of the graph in the DataWindow control.
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. For a list of line style values, see LineStyle on page 377.
linewidth	An integer variable in which you want to store the width of the line. The width is measured in pixels.

Return value

Returns 1 if it succeeds and -1 if an error occurs. For the specified series and data point, GetDataStyle stores its line style in *linestyle* and the line's width in *linewidth*. If any argument's value is NULL, GetDataStyle returns NULL.

Examples

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
GetDataStyleLineStyle
GetSeriesStyleLineWidth
GetSeriesStyle
SetDataStyle
SetSeriesStyle

Syntax 3 For the fill pattern of a data point

Description

Obtains the fill pattern of a data point in a graph.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

integer *dwcontrol*.**GetDataStyle** (string *graphcontrol*, integer *seriesnumber*, integer *datapointnumber*, REF FillPattern *fillvariable*)

Argument	Description
dwcontrol	A reference to the DataWindow control containing the graph.
graphcontrol	A string whose value is the name of the graph in the DataWindow control.
seriesnumber	The number of the series in which you want the fill pattern of a data point.
datapointnumber	The number of the data point for which you want the fill pattern.
fillvariable	A variable of type FillPattern in which you want to store the fill pattern value. For a list of values, see FillPattern on page 373.

Return value

Returns 1 if it succeeds and -1 if an error occurs. GetDataStyle stores a value of the FillPattern enumerated datatype representing the fill pattern used for the specified data point. If any argument's value is NULL, GetDataStyle returns NULL.

Examples

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)
```

See also

FindSeries
GetDataStyleFillPattern
GetSeriesStyle
SetDataStyle
SetSeriesStyle

Syntax 4 For the symbol of a data point

Description

Obtains the symbol of a data point in a graph.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

integer *dwcontrol*.**GetDataStyle** (string *graphcontrol*, integer *seriesnumber*, integer *datapointnumber*, REF grSymbolType *symbolvariable*)

Argument	Description
dwcontrol	A reference to the DataWindow control containing the graph.
graphcontrol	A string whose value is the name of the graph in the DataWindow control.
seriesnumber	The number of the series in which you want the symbol type of a data point.
datapointnumber	The number of the data point for which you want the symbol type.
symbolvariable	A variable of type grSymbolType in which you want to store the symbol type. For a list of values, see grSymbolType on page 376.

Return value

Returns 1 if it succeeds and -1 if an error occurs. Stores, according to the type of *symbolvariable*, a value of that enumerated datatype representing the symbol used for the specified data point. If any argument's value is NULL, GetDataStyle returns NULL.

Examples

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
GetDataStyleSymbolValue
GetSeriesStyle
SetDataStyle
SetSeriesStyle

GetDataStyleColorValue

Description

Returns the color value associated with a data point in a graph in a DataWindow object. You must call GetDataStyleColor first to retrieve the color information. (See GetDataStyle for information about this method.)

PocketBuilder	✗
PowerBuilder	✓

Syntax

Web ActiveX

number *dwcontrol*.**GetDataStyleColorValue** ( )

Return value

Returns an RGB color value.

GetDataStyleFillPattern

Description

Returns the fill pattern associated with a data point in a graph in a DataWindow object. You must call GetDataStyleFill first to retrieve the fill information. (See GetDataStyle for information about this method.)

PocketBuilder	✗
PowerBuilder	✓

Syntax

Web ActiveX

number *dwcontrol*.**GetDataStyleFillPattern** ( )

Return value

Returns an integer representing the fill pattern.

For a list of values and their meanings, see FillPattern on page 373.

GetDataStyleLineStyle

Description

Returns the line style associated with a data point in a graph in a DataWindow object. You must call GetDataStyleLine first to retrieve the line style information. (See GetDataStyle for information about this method.)

PocketBuilder	✗
PowerBuilder	✓

Syntax

Web ActiveX

number *dwcontrol*.**GetDataStyleLineStyle** ( )

Return value

Returns an integer representing the line style.

For a list of values and their meanings, see LineStyle on page 377.

GetDataStyleLineWidth

Description

Returns the line width associated with a data point in a graph in a DataWindow object. You must call GetDataStyleLine first to retrieve the line style information. (See GetDataStyle for information about this method.)

PocketBuilder	✗
PowerBuilder	✓

Syntax

Web ActiveX

number *dwcontrol*.**GetDataStyleLineWidth** ( )

Return value

Returns the width of the line in pixels.

GetDataStyleSymbolValue

Description

Returns the symbol associated with a data point in a graph in a DataWindow object. You must call GetDataStyleSymbol first to retrieve the symbol information. (See GetDataStyle for information about this method.)

PocketBuilder	✗
PowerBuilder	✓

Syntax

Web ActiveX

number *dwcontrol*.**GetDataStyleSymbolValue** ( )

Return value

Returns an integer representing data point's symbol. For a list of values and their meanings, see grSymbolType on page 376.

GetDataValue

Description

Obtains the value of a data point in a series in a graph.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

integer *dwcontrol*.**GetDataValue** (string *graphcontrol*, integer *seriesnumber*, long *datapoint*, REF date *datavariable*, { grDataType *XorY* })

integer *dwcontrol*.**GetDataValue** (string *graphcontrol*, integer *seriesnumber*, long *datapoint*, REF datetime *datavariable* {, grDataType *XorY* })

integer *dwcontrol*.**GetDataValue** (string *graphcontrol*, integer *seriesnumber*, long *datapoint*, REF double *datavariable* {, grDataType *XorY* })

integer *dwcontrol*.**GetDataValue** (string *graphcontrol*, integer *seriesnumber*, long *datapoint*, REF string *datavariable* {, grDataType *XorY* })

integer *dwcontrol*.**GetDataValue** (string *graphcontrol*, integer *seriesnumber*, long *datapoint*, REF time *datavariable* {, grDataType *XorY* })

Argument	Description
dwcontrol	A reference to the DataWindow control containing the graph.
graphcontrol	A string whose value is the name of the graph in the DataWindow control.
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 (optional argument, for scatter graph only)	A value of the grDataType enumerated datatype specifying whether you want the x or y value of the data point in a scatter graph. For values, see grDataType on page 375.

Return value

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, or returned by a method that corresponds to the axis datatype. If the values axis is numeric, you can also use the GetData function.

Calling GetDataValue when the datatype of *datavariable* is not the same as the datatype of the data produces undefined results.

If a variable's datatype is non-numeric and the datatype of *datavariable* is double, GetDataValue returns the number of the datapoint in *datavariable*.

If a variable's datatype is date, time, or DateTime, GetDataValue returns 1 when the datatype of *datavariable* is any of those datatypes. However, if the variable's datatype is time and the datatype of *datavariable* is date, GetDataValue returns 00/00/00 in *datavariable*, and if the variable's datatype is date and the datatype of *datavariable* is time, GetDataValue returns 00:00:00 in *datavariable*.

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)
```

See also

FindSeries
ObjectAtPointer

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 might 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 for the series	Syntax 3
Get the symbol for data points in the series	Syntax 4
Find out if the series is an overlay (a series shown as a line on top of another graph type)	Syntax 5

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	✓

Syntax

integer *dwcontrol*.**GetSeriesStyle** (string *graphcontrol*, string *seriesname*, grColorType *colortype*, REF long *colorvariable*)

Argument	Description
dwcontrol	A reference to the DataWindow control containing the graph.
graphcontrol	A string whose value is the name of the graph in the DataWindow control.
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. For a list of values, see grColorType on page 374.
colorvariable	A long variable in which you want to store the color's RGB value.

Return value

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 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)
```

See also

GetDataStyle
GetSeriesStyleColorValue

FindSeries
GetDataStyle
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	✓

Syntax

integer *dwcontrol*.**GetSeriesStyle** (string *graphcontrol*, string *seriesname*, REF LineStyle *linestyle* {, REF integer *linewidth* })

Argument	Description
dwcontrol	A reference to the DataWindow control containing the graph.
graphcontrol	A string whose value is the name of the graph in the DataWindow control.
seriesname	A string whose value is the name of the series for which you want the line style information.
linestyle	A variable of type LineStyle in which you want to store the line style of *seriesname*. For a list of values, see LineStyle on page 377.
linewidth (optional)	An integer variable in which you want to store the line width for *seriesname*. The width is measured in pixels.

Return value

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 = dw_equipment.ObjectAtPointer &
    ("gr_product_data", SeriesNbr, Data_Point)
```

```
IF MouseHit = TypeSeries! THEN
     SeriesName = &
          dw_equipment.SeriesName("gr_product_data", &
             SeriesNbr)

     dw_equipment.GetSeriesStyle ("gr_product_data", &
          SeriesName, line_style, line_width)
END IF
```

See also

GetDataStyle
GetDataStyleLineStyle
GetSeriesStyleLineWidth
FindSeries
SetDataStyle
SetSeriesStyle

Syntax 3 For the fill pattern of a series

Description

Obtains the fill pattern of a series in a graph.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

integer *dwcontrol*.**GetSeriesStyle** (string *graphcontrol*, string *seriesname*, REF FillPattern *fillvariable*)

Argument	Description
dwcontrol	A reference to the DataWindow control containing the graph.
graphcontrol	A string whose value is the name of the graph in the DataWindow control.
seriesname	A string whose value is the name of the series for which you want the style information.
fillvariable	A variable of type FillPattern in which you want to store the fill pattern value. For a list of values, see FillPattern on page 373.

Return value

Returns 1 if it succeeds and -1 if an error occurs. Stores in *fillvariable* a value identifying the fill pattern for the specified series. If any argument's value is NULL, GetSeriesStyle returns NULL.

Examples

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 = 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, data_pattern)

       gr_dept_data.SetSeriesStyle("Total Salary", &
          data_pattern)
END IF
```

See also

GetDataStyle
GetSeriesStyleFillPattern
FindSeries
SetDataStyle
SetSeriesStyle

Syntax 4 For the symbol of a series

Description

Obtains the symbol used for data points in a series in a graph.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

integer *dwcontrol*.**GetSeriesStyle** (string *graphcontrol*, string *seriesname*, REF grSymbolType *symbolvariable*)

Argument	Description
dwcontrol	A reference to the DataWindow control containing the graph.

Argument	Description
graphcontrol	A string whose value is the name of the graph in the DataWindow control.
seriesname	A string whose value is the name of the series for which you want the style information.
symbolvariable	A variable of type grSymbolType in which you want to store the symbol value. For a list of values, see grSymbolType on page 376.

Return value

Returns 1 if it succeeds and -1 if an error occurs. Stores in *symbolvariable* a value of the grSymbolType enumerated datatype for the symbol used for the specified series. If any argument's value is NULL, GetSeriesStyle returns NULL.

Examples

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
grSymbolType symbol
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, symbol)

       gr_dept_data.SetSeriesStyle("Total Salary", &
           symbol)
END IF
```

See also

GetDataStyle
GetSeriesStyleSymbolValue
FindSeries
SetDataStyle
SetSeriesStyle

Syntax 5 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	✓

Syntax

integer *dwcontrol*.**GetSeriesStyle** (string *graphcontrol*, string *seriesname*, REF boolean *overlayindicator*)

Argument	Description
dwcontrol	A reference to the DataWindow control containing the graph.
graphcontrol	A string whose value is the name of the graph in the DataWindow control.
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

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.

See also

GetSeriesStyleOverlayValue

GetSeriesStyleColorValue

Description

Returns the color value associated with a series in a graph in a DataWindow object. You must call GetSeriesStyleColor first to retrieve the color information. (See GetSeriesStyle for information about this method.)

PocketBuilder	✗
PowerBuilder	✓

Syntax

Web ActiveX

number *dwcontrol*.**GetSeriesStyleColorValue** ( )

Return value — Returns an RGB color value.

Usage — To find out the color associated with a series, call GetSeriesStyleColor to retrieve the information, then immediately afterward, call GetSeriesStyleColorValue and examine the return value.

Since data points in a series can have their own style settings, the color setting for a series might not match the color for a specific data point within that series.

See also — GetSeriesStyle

GetSeriesStyleFillPattern

Description — Returns the fill pattern associated with a series in a graph in a DataWindow object. You must call GetSeriesStyleFill first to retrieve the fill information. (See GetSeriesStyle for information about this method.)

PocketBuilder	✗
PowerBuilder	✓

Syntax — **Web ActiveX**

number *dwcontrol*.**GetSeriesStyleFillPattern** ( )

Return value — Returns an integer representing the fill pattern. For a list of values and their meanings, see FillPattern on page 373.

GetSeriesStyleLineStyle

Description — Returns the line style associated with a series in a graph in a DataWindow object. You must call GetSeriesStyleLine first to retrieve the line style information. (See GetSeriesStyle for information about this method.)

PocketBuilder	✗
PowerBuilder	✓

Syntax — **Web ActiveX**

number *dwcontrol*.**GetSeriesStyleLineStyle** ( )

Return value — Returns an integer representing the line style. For a list of possible values and their meanings, see LineStyle on page 377.

GetSeriesStyleLineWidth

Description

Returns the line width associated with a series in a graph in a DataWindow object. You must call GetSeriesStyleLine first to retrieve the line style information. (See GetSeriesStyle for information about this method.)

PocketBuilder	✗
PowerBuilder	✓

Syntax

Web ActiveX

number *dwcontrol*.**GetSeriesStyleLineWidth** ( )

Return value

Returns the width of the line in pixels.

GetSeriesStyleOverlayValue

Description

Returns a value indicating whether a series is an overlay, that is, whether it is shown on top of another graph type. You must call GetSeriesStyleOverlay first to retrieve the overlay information. (See GetSeriesStyle for information about this method.)

PocketBuilder	✗
PowerBuilder	✓

Syntax

Web ActiveX

boolean *dwcontrol*.**GetSeriesStyleOverlayValue** ( )

Return value

Returns true if the series is an overlay and false if it is not.

GetSeriesStyleSymbolValue

Description

Returns the symbol associated with a series in a graph in a DataWindow object. You must call GetSeriesStyleLine first to retrieve the line style information. (See GetSeriesStyle for information about this method.)

PocketBuilder	✗
PowerBuilder	✓

Syntax

Web ActiveX

number *dwcontrol*.**GetSeriesStyleSymbolValue** ( )

Return value

Returns an integer representing a data point's symbol. For a list of values and their meanings, see grSymbolType on page 376.

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	✓

Syntax

grObjectType *dwcontrol*.**ObjectAtPointer** (string *graphcontrol*, REF integer *seriesnumber*, REF integer *datapoint*)

Return value

Returns a value of the grObjectType enumerated datatype identifying the type of object under the pointer 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.

ObjectAtPointerDataPoint

Description

Returns the number of the data point under the pointer. You must call ObjectAtPointer first to retrieve the pointer position information.

PocketBuilder	✗
PowerBuilder	✓

Syntax

Web ActiveX

number *dwcontrol*.**ObjectAtPointerDataPoint** ( )

Return value

Returns the number of the data point.

ObjectAtPointerSeries

Description

Returns the number of the series under the pointer. You must call ObjectAtPointer first to retrieve the pointer position information.

PocketBuilder	✗
PowerBuilder	✓

Syntax

Web ActiveX

number *dwcontrol*.**ObjectAtPointerSeries** ( )

Return value

Returns the number of the series.

Reset

Description

Deletes the data, the categories, or the series from a graph.

Reset is for graphs within a DataWindow object with an external data source. It does not apply to other graphs in DataWindow objects because their data comes directly from the DataWindow.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

integer *dwcontrol*.**Reset** (grResetType *graphresettype*)

Argument	Description
dwcontrol	A reference to the DataWindow control or DataStore containing the graph.
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 *dwcontrol*. • Category! — Delete categories and data in *dwcontrol*. • Data! — Delete data in *dwcontrol*. • Series! — Delete the series and data in *dwcontrol*.

Return value

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 in the DataWindow dw_prod. The DataWindow object has an external data source:

```
dw_prod.Reset("gr_product_data", Series!)
```

ResetDataColors

Description

Restores the color of a data point to the default color for its series.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

integer *dwcontrol*.**ResetDataColors** (string *graphcontrol*, integer *seriesnumber*, long *datapointnumber*)

Argument	Description
dwcontrol	A reference to the DataWindow control containing the graph
graphcontrol	A string whose value is the name of the graph in the DataWindow control
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

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_computers in the DataWindow control dw_equipment to the color for the series:

```
SeriesNbr = dw_equipment.FindSeries("gr_computers", &
      "Costs")
```

```
dw_equipment.ResetDataColors("gr_computers", &
        SeriesNbr, 10)
```

See also

GetDataStyle
GetSeriesStyle
SetDataStyle
SetSeriesStyle

SaveAs

Description

Saves the data in a graph in the format you specify.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

integer *dwcontrol*.**SaveAs** (string *graphcontrol* {, string *filename*, SaveAsType *saveastype*, boolean *colheading* })

Argument	Description
dwcontrol	A reference to the DataWindow control or DataStore containing the graph.
graphcontrol	A string whose value is the name of the graph in the DataWindow control or DataStore.
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 (""), the user is prompted for a file name.
saveastype (optional)	A value of the SaveAsType enumerated datatype specifying the format in which to save the data represented in the graph. For a list of values, see SaveAsType on page 378.
colheading (optional)	A boolean value indicating whether you want column headings with the saved data. The default value is TRUE. This argument is used for the following formats: Clipboard, CSV, Excel, Excel5, and Text. For most other formats, column headings are always saved.

Return value

Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is NULL, SaveAs returns NULL.

Usage

If you do not specify any arguments, PocketBuilder saves the DataWindow data rather than the data in the graph control. In this case, or in the case where you specify only the graph control name as an argument, PocketBuilder displays the Save As dialog box, letting the user specify the format of the saved data.

Examples

This statement saves the contents of gr_computers in the DataWindow control dw_equipmt to the file G:\INVENTORY\SALES.XLS. The format is comma-separated values with column headings:

```
dw_equipmt.SaveAs("gr_computers", &
       "G:\INVENTORY\SALES.XLS", CSV!, TRUE)
```

See also

Print
SaveAs

SeriesCount

Description

Counts the number of series in a graph.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

integer *dwcontrol*.**SeriesCount** (string *graphcontrol*)

Argument	Description
dwcontrol	A reference to the DataWindow control containing the graph
graphcontrol	A string whose value is the name of the graph in the DataWindow control

Return value

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_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	✓

Syntax

integer *dwcontrol*.**SeriesName** (string *graphcontrol*, integer *seriesnumber*)

Argument	Description
dwcontrol	A reference to the DataWindow control containing the graph
graphcontrol	A string whose value is the name of the graph in the DataWindow control
seriesnumber	The number of the series for which you want to obtain the name

Return value

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_computers in the DataWindow control dw_equipment:

```
string ls_SeriesName
ls_SeriesName = &
       dw_equipment.SeriesName("gr_computers", 5)
```

See also

CategoryName
GetData

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	✓

Syntax

integer *dwcontrol*.**SetDataPieExplode** (string *graphcontrol*, integer *seriesnumber*, integer *datapoint*, integer *percentage*)

Argument	Description
dwcontrol	A reference to the DataWindow control containing the graph.
graphcontrol	A string whose value is the name of the graph in the DataWindow control.
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 that 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

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, SetDataPieExplode 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 the DataWindow control:

```
integer series, datapoint
grObjectType clickedtype
integer percentage

percentage = 50
IF (This.GraphType <> PieGraph! AND &
       This.GraphType <> Pie3D!) THEN RETURN
```

```
clickedtype = This.ObjectAtPointer( "gr_equipment", &
       series, datapoint)

IF (series > 0 and datapoint > 0) THEN
       This.SetDataPieExplode("gr_equipment", 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 for the data point	Syntax 3
Set the symbol for the data point	Syntax 4

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	✓

Syntax integer *dwcontrol*.**SetDataStyle** (string *graphcontrol*, integer *seriesnumber*, integer *datapointnumber*, grColorType *colortype*, long *color*)

Argument	Description
dwcontrol	A reference to the DataWindow control containing the graph.
graphcontrol	A string whose value is the name of the graph in the DataWindow control.
seriesnumber	The number of the series in which you want to set the color of a data point.

Argument	Description
datapointnumber	The number of the data point for which you want to set the color.
colortype	A value of the grColorType enumerated datatype specifying the aspect of the data point for which you want to set the color. For a list of values, see grColorType on page 374.
color	A long whose value is the new color for *colortype*.

Return value

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.

You can specify the appearance of a data point in the graph before the application 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

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
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	✓

Syntax

integer *dwcontrol*.**SetDataStyle** (string *graphcontrol*, integer *seriesnumber*, integer *datapointnumber*, LineStyle *linestyle*, { integer *linewidth* })

Argument	Description
dwcontrol	A reference to the DataWindow control containing the graph.
graphcontrol	A string whose value is the name of the graph in the DataWindow control.
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.
linestyle	A value of the LineStyle enumerated datatype specifying a line style pattern of dots, dashes, and solid lines. For a list of line style values, see LineStyle on page 377.
linewidth (optional)	An integer whose value is the width of the line in pixels.

Return value

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.

You can specify the appearance of a data point in the graph before the application 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
SetSeriesStyle

Syntax 3 For the fill pattern of a data point

Description

Specifies the fill pattern for a data point in a graph.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

integer *dwcontrol*.**SetDataStyle** (string *graphcontrol*, integer *seriesnumber*, integer *datapointnumber*, FillPattern *fillvalue*)

Argument	Description
dwcontrol	A reference to the DataWindow control containing the graph.
graphcontrol	A string whose value is the name of the graph in the DataWindow control.
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.
fillvalue	A value of the FillPattern enumerated datatype specifying the fill pattern for the data point. For a list of values, see FillPattern on page 373.

Return value

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.

You can specify the appearance of a data point in the graph before the application 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.

See also

GetDataStyle
GetSeriesStyle
SetSeriesStyle

Syntax 4 For the symbol of a data point

Description

Specifies the symbol for a data point in a graph.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

integer *dwcontrol*.**SetDataStyle** (string *graphcontrol*, integer *seriesnumber*, integer *datapointnumber*, grSymbolType *symbolvalue*)

Argument	Description
dwcontrol	A reference to the DataWindow control containing the graph.
graphcontrol	A string whose value is the name of the graph in the DataWindow control.
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.
symbolvalue	A value of the grSymbolType enumerated datatype specifying the symbol for the data point. For a list of values, see grSymbolType on page 376.

Return value

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.

You can specify the appearance of a data point in the graph before the application 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.

See also

GetDataStyle
GetSeriesStyle
SetSeriesStyle

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 for the series	Syntax 3
Set the symbol for the series	Syntax 4
Specify that the series is an overlay	Syntax 5

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	✓

Syntax

integer *dwcontrol*.**SetSeriesStyle** (string *graphcontrol*, string *seriesname*, grColorType *colortype*, long *color*)

Argument	Description
dwcontrol	A reference to the DataWindow control containing the graph.
graphcontrol	A string whose value is the name of the graph in the DataWindow control.

Argument	Description
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. For a list of values, see grColorType on page 374.
color	A long specifying an RGB value for the new color.

Return value

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.

You can specify the appearance of a series in the graph before the application 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 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 within the DataWindowcontrol dw_sales to that color:

```
string SeriesName
integer SeriesNbr, Series_Point
long line_color
grObjectType MouseHit
MouseHit = This.ObjectAtPointer( &
       SeriesNbr,Series_Point)

IF MouseHit = TypeSeries! THEN
       SeriesName = &
          gr_product_data.SeriesName(SeriesNbr)
```

```
            gr_product_data.GetSeriesStyle(SeriesName, &
               LineColor!, line_color)

            dw_sales.SetSeriesStyle("gr_sales_data", &
               "Northeast", LineColor!, line_color)
   END IF
```

See also

GetDataStyle
GetSeriesStyle
SetSeriesStyle

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	✓

Syntax

integer *dwcontrol*.**SetSeriesStyle** (string *graphcontrol*, string *seriesname*, LineStyle *linestyle* {, integer *linewidth* })

Argument	Description
dwcontrol	A reference to the DataWindow control containing the graph.
graphcontrol	A string whose value is the name of the graph in the DataWindow control.
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 specifying the line style. For a list of values, see LineStyle on page 377.
linewidth (optional)	An integer specifying the width of the line in pixels.

Return value

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.

You can specify the appearance of a series in the graph before the application 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 in the DataWindow dw_prod:

```
dw_prod.SetSeriesStyle("gr_product_data", "Costs", &
       Dot!, 5)
```

See also

GetDataStyle
GetSeriesStyle
SetDataStyle

Syntax 3 For the fill pattern in a graph

Description

Specifies the fill pattern for data markers in a series.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

integer *dwcontrol*.**SetSeriesStyle** (string *graphcontrol*, string *seriesname*, FillPattern *fillvalue*)

Argument	Description
dwcontrol	A reference to the DataWindow control containing the graph.
graphcontrol	A string whose value is the name of the graph in the DataWindow control.
seriesname	A string whose value is the name of the series in which you want to set the appearance.
fillvalue	A value of the FillPattern enumerated datatype specifying the fill pattern for the series. For a list of values, see FillPattern on page 373.

Return value

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.

You can specify the appearance of a series in the graph before the application 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 fill pattern used for the series named Costs in the graph gr_computers in the DataWindow control dw_equipment to Horizontal:

```
dw_equipment.SetSeriesStyle("gr_computers", &
        "Costs", Horizontal!)
```

See also

GetDataStyle
GetSeriesStyle
SetDataStyle

Syntax 4 For the symbols in a graph

Description

Specifies the symbol for data markers in a series.

PocketBuilder on Pocket PC	✓
PocketBuilder on Smartphone	✓
PowerBuilder	✓

Syntax

integer *dwcontrol*.**SetSeriesStyle** (string *graphcontrol*, string *seriesname*, grSymbolType *symbolvalue*)

Argument	Description
dwcontrol	A reference to the DataWindow control containing the graph.
graphcontrol	A string whose value is the name of the graph in the DataWindow control.
seriesname	A string whose value is the name of the series in which you want to set the appearance.
symbolvalue	A value of the grSymbolType enumerated datatype specifying the symbol for the series. For a list of values, see grSymbolType on page 376.

Return value

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.

You can specify the appearance of a series in the graph before the application 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 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
SetDataStyle

Syntax 5 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	✓

Syntax

integer *dwcontrol*.**SetSeriesStyle** (string *graphcontrol*, string *series*, boolean *overlaystyle*)

Argument	Description
dwcontrol	A reference to the DataWindow control containing the graph.
graphcontrol	A string whose value is the name of the graph in the DataWindow control.
series	A string whose value is the name of the series whose overlay status you want to change.

Argument	Description
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

Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is NULL, SetSeriesStyle returns NULL.

Usage

You can specify the appearance of a series in the graph before the application 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

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
SetDataStyle

Index

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z