PUBLIC

PowerBuilder 12.6
Document Version: 1.0 - 2014-08-07

New Features

Table of Contents

1 New Features

Significant features introduced in the 12.6 release of PowerBuilder Classic and PowerBuilder .NET.

2 OData Support

PowerBuilder Classic and PowerBuilder .NET can use OData datasources.

OData (the Open Data Protocol) is a Web protocol for querying and updating data that provides a way to unlock your data and free it from silos that exist in applications. OData is based on REST (Representational State Transfer). It uses the standard HTTP protocol to access data using GET, PUT, POST, and DELETE.

At runtime, users can retrieve and manipulate data.

2.1 Database Profile Setup - OData Dialog Box

Define a database profile to access an OData service in PowerBuilder using the OData interface.

2.1.1 Connection Tab

The Connection tab includes basic connection options that you must supply to access the information in the OData service.

Profile Name	The name of your database profile.
Connection Information	The uniform resource identifier (URI) that represents the OData service.
Authenticated Access	Select the type of access: • Anonymous access – Opens up the service so it can be accessed without providing credentials. • Integrated Windows authentication – Uses information on the client computer to validate the user's access. • Supply user ID and password – If you select this option, enter the user name required to connect to the datasource and the password for the specified login ID.
OData Extended Catalog	Use this option to define the extended attributes, such as edit styles and validation rules, that can be applied in columns of the tables. After you select this option, you can also specify the properties for each table and column. This information is saved in the registry.

2.1.2 Certificate Tab

The Certificate tab includes basic connection options that you must supply to access the information in the OData service.

Select one connection option:

No X509 Certificate	Select this option when the OData service does not require a certificate.
Select a certificate from current user's personal store	Click *Change* to select a certificate.
Specify a local certificate file	Click *Browse* to select a certificate file on the local machine.

2.1.3 Proxy Server Tab

The Proxy Server tab has additional connection options that you can use to manage access to the OData service.

Use proxy server settings in Tools \ Options dialog	Select this option when you need proxy or firewall settings to access the OData services. Before you select this option, set the proxy or firewall information in *Tools* > *System Options* > *Firewall Setting*. Be sure to select Use Above Values as System Defaults.
Bypass proxy server for local addresses	Use this option to bypass the proxy server when the OData Service is using a local address.

2.1.4 Preview Tab

The Preview tab provides a convenient way to generate correct PowerScript connection syntax in the PowerBuilder development environment for use in your PowerBuilder application script.

As you complete the Database Profile Setup dialog box for OData, the correct PowerScript connection syntax for each select option is built on the Preview tab. You can then copy the syntax you want from the Preview tab into your PowerBuilder application script.

Copy	Copies the selected text in the Database Connection Syntax box to the clipboard. You can then paste the syntax into your PowerBuilder script.

2.2 Database Painter

You can connect to the OData service and work with it in the Database painter.

The Database painter supports these features:

- Define an OData datasource with the database profile painter.
- Connect to an OData datasource in the Objects view.
- Add tables to the Object Layout view with drag-and-drop or the context menu.
- Display table and column properties in the Properties view with drag-and-drop or the context menu.
- Drag and drop a table onto the Columns view.

- Invoke the Edit Data feature in the Results view of the Database painter.

Since OData is not a SQL source, some features are not supported:

- The ISQL Session view is not enabled.
- You cannot create, modify, or delete tables.
- You cannot create users or groups of users.
- You cannot embed SQL statements in PowerScript.

2.3 Create a DataWindow Using an OData Service

Select the OData Service datasource in the DataWindow wizard.

Context

Procedure

1. Select *File* > *New* from the menu bar and select DataWindow.
2. If there is more than one target, select the target where you want the DataWindow to be created from the drop-down list.
3. Choose the presentation style for the DataWindow object and click *Next*.
4. Select the *OData Service* datasource and click *Next*.
5. Select the OData profile and click *Next*.
6. In the SQL painter:
   - You can select one table.
   - The Sort, Group, and Having tabs are not available.
   - The Where and Syntax are available in PowerBuilder Classic and PowerBuilder .NET. The Results tab is available in PowerBuilder .NET.
   - In the Where tab you can specify some selection criteria using the WHERE clause for the SELECT statement.

   When you complete the query, click *OK*.
7. Review your specifications and click *Finish*.

Results

At runtime, the DataWindow or DataStore can manipulate OData service data, which includes retrieving, updating, inserting, and deleting the data.

2.4 Set the Connection Information for the OData Service

As with other databases, use the SQLCA Transaction object (or user-defined transaction object) to retrieve and display data from the OData service in a DataWindow or DataStore.

Procedure

1. Set the appropriate values for the transaction object.
2. Connect to the OData service.
3. Set the transaction object for the DataWindow or DataStore.
4. Retrieve and update the data.
5. When the processes are complete, disconnect from the OData service.

Example

The code looks something like this:

```
SQLCA.DBMS = "ODT"
SQLCA.DBParm = "ConnectString='URI=http://esx2-appserver/TestDataService/
Employee.svc'"

//connect to the service
connect using SQLCA;

dw_1.SetTransObject(SQLCA)
dw_1.Retrieve()

...

//disconnect from the service
disconnect using SQLCA;
```

For more information on using the global Transaction object, see *Application Techniques*.

3 64-Bit Windows Applications

Create 64-bit native applications in PowerBuilder Classic.

Usage

There is no special target for 64-bit native applications. To build a 64-bit application, select the platform in the Project painter General tab. If you need to deliver both 32-bit and 64-bit versions of your application, you should use separate projects and separate folders for the deployed output.

There is no IDE for 64-bit development. Design time uses the same 32-bit interface and 64-bit features display at runtime when you deploy the application. When you click the running man button, the project runs as a 32-bit application.

32-bit remains the default for new and migrated applications.

During the deploy process, PowerBuilder checks and reports unsupported features used in the application.

New Property for Environment Object

The new `ProcessBitness` property identifies whether the application is a 32-bit or 64-bit process.

Datatype	integer
Values	32 stands for 32-bit, and 64 stands for 64-bit

See *Objects and Controls* for more about the `Environment` object. See the *PowerScript Reference* to read about the `GetEnvironment` function.

New Datatype

The `longptr` datatype is 4 bytes in the 32-bit platform and 8 bytes in the 64-bit platform. In the 32-bit platform, `longptr` is the same as `long`; you can continue using long wherever `longptr` is required in 32-bit applications. In 64-bit applications, however, using `long` to hold `longptr` variables will lead to data truncation from 8 bytes to 4 bytes, or memory corruption if you pass a

```
long
```

ref variable when a `longptr` ref is required. If you want to move to 64-bit, use `longptr` wherever required. It does no harm to 32-bit.

Since PowerBuilder does not have a datatype corresponding to the C++ pointer type, and there are no pointer operations in PowerBuilder, `longptr` is not a full-fledged PowerBuilder datatype. You can use it to hold/pass window handles, database handles, and other objects that are essentially memory addresses. Doing complex

operations on `longptr` type might not work. If you want to represent/compute 8-byte long integers, use `longlong`.

System Requirements

The design time environment requires:

- Windows SDK for Windows 7 or later
- .NET Framework 4.0 or later
- 64-bit Windows OS to test (development requires only 32-bit)

The runtime environment requires:

- 64-bit Windows OS
- PowerBuilder 12.6 64-bit system files
- 64-bit third-party libraries, such as database drivers and external DLLs
- Greater than 4 GB physical memory to avoid performance issues

Limitations

There are limitations to this new feature:

- To consume Web services, you must use the .NET engine. EasySOAP is not supported.
- You can use OLE and ActiveX components in your applications, but you must use the 32-bit versions in the PowerBuilder Classic IDE. At runtime you must have the correct 64-bit ActveX components installed.
- The RichText DataWindow header does not display when the HeaderFooter property is true until you call `ShowHeaderFooter(true)`. If you do not:
  - `selecttext ( long l1, long c1, long l2, long c2, band b Header! )` returns 0 and selected text is '' (string with 0 length)
  - `selecttext ( long l1, long c1, long l2, long c2, band b Footer! )` returns 0 and selected text is '' (string with 0 length)

  Regardless, autofocus does not work.
- Scrolling in a RichText DataWindow loses focus.
- CopyRTF(false,header!) works only when you call ShowHeaderFooter(true) when Header/Footer is true
-
- InsertDocument("*.htm",true) returns -1
- InsertDocument("*.doc",true) returns -1
-
- Position returns the header when the footer is in focus
- SaveDocument (string f, {FileTypeDoc!|FileTypeHTML!|FileTypePDF!}) retuns -1 and FileExists event is triggered

Unsupported Features

These features are not supported:

- COM+ runtime
- Machine code generation
- TabletPC
- PBNI SDK for developing 64-bit PowerBuilder extensions
- DataWindow RichText style column
- DataWindow Web control for ActiveX
- Status bar
- Grid table
- ClearAll() function
- Clear(true) function
- Change Pointer does not work on RichTextEdit controls
- Mouse wheel does not scroll a RichTextEdit page
- Application server support

Also, if you select Properties in the RichTextEdit Object Dialog popup menu, the application crashes if you select the Print Spec tabpage and click OK.

Behavior Differences

Some things are not problematic, simply different:

- The RichText preview mode behaves differently; in 64-bit, it is more like a print preview

PowerBuilder Native Interface (PBNI)

You can only use 32-bit PowerBuilder extensions in the PowerBuilder Classic IDE. For runtime, package and distribute 64-bit extension libraries with your 64-bit applications. The file names of your 64-bit extension should match the 32-bit file names, since the application references it by file name.

OrcaScript

To build 64-bit native applications with OrcaScript, use the new X64 option to build executable commands. For example:

```
build executable exeNameiconNamepbrNamepbdflags x64
```

4 Dockable Windows

In PowerBuilder Classic, you can set docking behaviors for the sheets that open in the MDI (multiple document interface) frame window.

4.1 Window Types and Docking States

New window types allow sheets to open in one of four states: docked, floating, tabbed document, or tabbed window.

The two new `WindowType` values are `mdidock!` and `mdidockhelp!`. Like `mdi!` and `mdihelp!`, you can open sheets (child windows) with the new OpenSheet functions.

`Docked`	The sheet is open and fixed in position relative to the Window object. The docked state is the default.
`Floating`	Users can move a floating sheet around or even outside the Window object.
`TabbedDocument`	Sheets that appear tabbed in the same area of the Window.
`TabbedWindow`	Docked windows that occupy the same area of the Window are in a tabbed group. The tabs are at the bottom.

You can get each opened sheet's status using the enumerated type WindowDockState.

- `WindowDockStateDocked!`
- `WindowDockStateFloating!`
- `WindowDockStateTabbedDocument!`
- `WindowDockStateTabbedWindow!`

4.2 Open Sheets in a Specific State

New versions of the `OpenSheet` function allow you to open a sheet at a specific docking location, in a specific tab group, or as a document.

You can programmatically open sheets in a specific state using these new PowerScript® functions:

- OpenSheetAsDocument
- OpenSheetDocked
- OpenSheetInTabGroup
- OpenSheetWithParmAsDocument
- OpenSheetWithParmDocked
- OpenSheetWithParmInTabGroup

4.2.1 OpenSheetAsDocument PowerScript function

Opens a sheet as a document within an MDI frame window for dockable windows.

Applies to

Windows objects

Syntax

```
OpenSheetAsDocument ( <sheetrefvar> {, <windowtype> }, <mdiframe>, <sheetname>
{, <tabalign> } )
```

Argument	Description
<sheetrefvar>	The name of any window variable that is not an MDI frame window. `OpenSheetAsDocument` places a reference to the open sheet in **<sheetrefvar>**.
<windowtype> (optional)	A string whose value is the datatype of the window you want to open. The datatype of **<windowtype>** must be the same or a descendant of **<sheetrefvar>**.
<mdiframe>	The name of an MDI frame window.
<sheetname>	A unique string identifier for the sheet, which is used when layout is persisted.
<tabalign> (optional)	A boolean that, when used, creates a new tab group and indicates the alignment of the sheets in the group. When true, the tabs in the group align vertically. When false, the tabs align horizontally.

Returns

Integer. Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is null, `OpenSheetAsDocument` returns null. In some cases, such as if the **<windowtype>** argument is invalid, `OpenSheetAsDocument` throws a runtime error and does not return a value; therefore, it is recommended that you both test the return value and wrap the function call in a try-catch block.

Usage

Tabbed documents can be in more than one tab group. Users can create additional tab groups by dragging one tab outside of the current group. If there is more than one tab group, use the **<mdiframe>** argument to specify in which one to open a sheet. Instead of specifying the parent window, specify an already open sheet in the tab group where you want to open your new sheet.

4.2.2 OpenSheetWithParmAsDocument PowerScript function

Opens a sheet as a document within an MDI frame window for dockable windows. `OpenSheetWithParmAsDocument` also stores a parameter in the system's Message object so that it is accessible to the opened sheet.

Applies to

Windows objects

Syntax

```
OpenSheetWithParmAsDocument ( <sheetrefvar>, <parameter> {, <windowtype> },
<mdiframe>, <sheetname> {, <tabalign> } )
```

Argument	Description
<sheetrefvar>	The name of any window variable that is not an MDI frame window. `OpenSheetWithParmAsDocument` places a reference to the open sheet in **<sheetrefvar>**.
<parameter>	The parameter you want to store in the Message object when the sheet is opened. **<Parameter>** must have one of these datatypes: • String • Double • PowerObject
<windowtype> (optional)	A string whose value is the datatype of the window you want to open. The datatype of **<windowtype>** must be the same or a descendant of **<sheetrefvar>**.

Argument	Description
<mdiframe>	The name of an MDI frame window.
<sheetname>	A unique string identifier for the sheet, which is used when layout is persisted.
<tabalign> (optional)	A boolean that, when used, creates a new tab group and indicates the alignment of the sheets in the group. When true, the tabs in the group align vertically. When false, the tabs align horizontally.

Returns

Integer. Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is null, `OpenSheetWithParmAsDocument` returns null. In some cases, such as if the **<windowtype>** argument is invalid, `OpenSheetWithParmAsDocument` throws a runtime error and does not return a value; therefore, it is recommended that you both test the return value and wrap the function call in a try-catch block.

Usage

Tabbed documents can be in more than one tab group. Users can create additional tab groups by dragging one tab outside of the current group. If there is more than one tab group, use the **<mdiframe>** argument to specify in which one to open a sheet. Instead of specifying the parent window, specify an already open sheet in the tab group where you want to open your new sheet.

The system Message object has three properties for storing data. Depending on the datatype of the parameter specified for `OpenSheetWithParmAsDocument`, scripts for the opened sheet would check one of the following properties.

Message object property	Argument datatype
Message.DoubleParm	Double
Message.PowerObjectParm	PowerObject (PowerBuilder objects, including user-defined structures)
Message.StringParm	String

In the opened window, it is a good idea to access the value passed in the Message object immediately (because some other script may use the Message object for another purpose).

> **Note**
>
> When you pass a PowerObject as a parameter, you are passing a reference to the object. The object must exist when you refer to it later or you get a null object reference, which causes an error. For example, if you pass the name of a control on a window that is being closed, that control will not exist when a script accesses the parameter.

4.2.3 OpenSheetDocked PowerScript function

Opens a sheet docked in a specified position within an MDI frame window for dockable windows.

Applies to

Windows objects

Syntax

```
OpenSheetDocked ( <sheetrefvar> {, <windowtype> }, <mdiframe>, <position>,
<sheetname> )
```

Argument	Description
<sheetrefvar>	The name of any window variable that is not an MDI frame window. `OpenSheetDocked` places a reference to the open sheet in **<sheetrefvar>**.
<windowtype> (optional)	A string whose value is the datatype of the window you want to open. The datatype of **<windowtype>** must be the same or a descendant of **<sheetrefvar>**.
<mdiframe>	The name of an MDI frame window.
<position>	An enumerated type that specifies where to dock the sheet: • WindowDockLeft! • WindowDockRight! • WindowDockTop! • WindowDockBottom!
<sheetname>	A unique string identifier for the sheet, which is used when layout is persisted.

Returns

Integer. Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is null, `OpenSheetDocked` returns null. In some cases, such as if the **<windowtype>** argument is invalid, `OpenSheetDocked` throws a runtime error and does not return a value; therefore, it is recommended that you both test the return value and wrap the function call in a try-catch block.

Usage

Open the sheet, docked in a specified position.

4.2.4 OpenSheetWithParmDocked PowerScript function

Opens a sheet docked in a specified position within an MDI frame window for dockable windows. `OpenSheetWithParmDocked` also stores a parameter in the system's Message object so that it is accessible to the opened sheet.

Applies to

Windows objects

Syntax

```
OpenSheetWithParmDocked ( <sheetrefvar>, <parameter> {, <windowtype> },
<mdiframe>, <position>, <sheetname> )
```

Argument	Description
<sheetrefvar>	The name of any window variable that is not an MDI frame window. `OpenSheetWithParmDocked` places a reference to the open sheet in **<sheetrefvar>**.
<parameter>	The parameter you want to store in the Message object when the sheet is opened. **<Parameter>** must have one of these datatypes: • String • Double • PowerObject
<windowtype> (optional)	A string whose value is the datatype of the window you want to open. The datatype of **<windowtype>** must be the same or a descendant of **<sheetrefvar>**.
<mdiframe>	The name of an MDI frame window.
<position>	An enumerated type that specifies where to dock the sheet: • WindowDockLeft!

Argument	Description
	• WindowDockRight! • WindowDockTop! • WindowDockBottom!
<sheetname>	A unique string identifier for the sheet, which is used when layout is persisted.

Returns

Integer. Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is null, `OpenSheetWithParmDocked` returns null. In some cases, such as if the **<windowtype>** argument is invalid, `OpenSheetWithParmDocked` throws a runtime error and does not return a value; therefore, it is recommended that you both test the return value and wrap the function call in a try-catch block.

Usage

The system Message object has three properties for storing data. Depending on the datatype of the parameter specified for `OpenSheetWithParmFromDocked`, scripts for the opened sheet would check one of the following properties.

Message object property	Argument datatype
Message.DoubleParm	Double
Message.PowerObjectParm	PowerObject (PowerBuilder objects, including user-defined structures)
Message.StringParm	String

In the opened window, it is a good idea to access the value passed in the Message object immediately (because some other script may use the Message object for another purpose).

> **Note**
>
> When you pass a PowerObject as a parameter, you are passing a reference to the object. The object must exist when you refer to it later or you get a null object reference, which causes an error. For example, if you pass the name of a control on a window that is being closed, that control will not exist when a script accesses the parameter.

4.2.5 OpenSheetInTabGroup PowerScript function

Opens a sheet in a tab group within an MDI frame window for dockable windows.

Applies to

Windows objects

Syntax

```
OpenSheetInTabGroup ( <sheetrefvar> {, <windowtype> }, <siblingname>,
<sheetname> )
```

Argument	Description
<sheetrefvar>	The name of any window variable that is not an MDI frame window. `OpenSheetInTabGroup` places a reference to the open sheet in **<sheetrefvar>**.
<windowtype> (optional)	A string whose value is the datatype of the window you want to open. The datatype of **<windowtype>** must be the same or a descendant of **<sheetrefvar>**.
<siblingname>	The name of a sibling window in either a docked state or in a non-document tab group. The sheet opens in that tab group.
<sheetname>	A unique string identifier for the sheet, which is used when layout is persisted.

Returns

Integer. Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is null, `OpenSheetInTabGroup` returns null. In some cases, such as if the **<windowtype>** argument is invalid, `OpenSheetInTabGroup` throws a runtime error and does not return a value; therefore, it is recommended that you both test the return value and wrap the function call in a try-catch block.

Usage

The first sheet opened in a main window cannot be opened using `OpenSheetInTabGroup` or `OpenSheetWithParmInTabGroup`. To create a tab group, open the first sheet as a docked sheet and then use that sheet as the **<siblingname>** argument.

4.2.6 OpenSheetWithParmInTabGroup PowerScript function

Opens a sheet in a tab group within an MDI frame window for dockable windows. `OpenSheetWithParmInTabGroup` also stores a parameter in the system's Message object so that it is accessible to the opened sheet.

Applies to

Windows objects

Syntax

```
OpenSheetWithParmInTabGroup ( <sheetrefvar>, <parameter> {, <windowtype> },
<siblingname>, <sheetname> )
```

Argument	Description
<sheetrefvar>	The name of any window variable that is not an MDI frame window. `OpenSheetWithParmInTabGroup` places a reference to the open sheet in **<sheetrefvar>**.
<parameter>	The parameter you want to store in the Message object when the sheet is opened. **<Parameter>** must have one of these datatypes: • String • Double • PowerObject
<windowtype> (optional)	A string whose value is the datatype of the window you want to open. The datatype of **<windowtype>** must be the same or a descendant of **<sheetrefvar>**.

Argument	Description
<siblingname>	The name of a sibling window in either a docked state or in a non-document tab group. The sheet opens in that tab group.
<sheetname>	A unique string identifier for the sheet, which is used when layout is persisted.

Returns

Integer. Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is null, `OpenSheetWithParmInTabGroup` returns null. In some cases, such as if the **<windowtype>** argument is invalid, `OpenSheetWithParmInTabGroup` throws a runtime error and does not return a value; therefore, it is recommended that you both test the return value and wrap the function call in a try-catch block.

Usage

The first sheet opened in a main window cannot be opened using `OpenSheetInTabGroup` or `OpenSheetWithParmInTabGroup`. To create a tab group, open the first sheet as a docked sheet and then use that sheet as the **<siblingname>** argument.

The system Message object has three properties for storing data. Depending on the datatype of the parameter specified for `OpenSheetWithParmFromDockingState`, scripts for the opened sheet would check one of the following properties.

Message object property	Argument datatype
Message.DoubleParm	Double
Message.PowerObjectParm	PowerObject (PowerBuilder objects, including user-defined structures)
Message.StringParm	String

In the opened window, it is a good idea to access the value passed in the Message object immediately (because some other script may use the Message object for another purpose).

> **Note**
>
> When you pass a PowerObject as a parameter, you are passing a reference to the object. The object must exist when you refer to it later or you get a null object reference, which causes an error. For example, if you pass the name of a control on a window that is being closed, that control will not exist when a script accesses the parameter.

4.2.7 Opening Docked Windows and Tabbed Document Windows

Sample code for opening docked windows and tabbed documents.

Context

Procedure

1. Create a window w_sheet_any as a main! window type.
2. In the open event of w_sheet_any, add this code:

```
string ls_i
ls_i = Message.stringparm
if not isnull(ls_i) and ls_i <> "" then
    this.title = ls_i
end if
```

3. Create an MDIDock window w_mdidock_dockstate and set any menu in it.
4. In the open event of w_mdidock_dockstate, add this code:

```
window win[]
OpenSheetWithParmDocked(win[1], "1", "w_sheet_any", this, WindowDockLeft!, "")
OpenSheetWithParmInTabGroup(win[2], "2", "w_sheet_any", this, "")
OpensheetWithParmInTabGroup(win[3], "3", "w_sheet_any", win[1], "")
OpenSheetWithParmAsDocument(win[4], "4", "w_sheet_any", this, "")
OpenSheetWithParmAsDocument(win[5], "5", "w_sheet_any", win[4], "")
OpenSheetWithParmAsDocument(win[6], "6", "w_sheet_any", this, "", false)
OpenSheetWithParmAsDocument(win[7], "7", "w_sheet_any", win[5],  "")
OpenSheetWithParmAsDocument(win[8], "8", "w_sheet_any", win[6],  "")
```

5. Run the application.
   You will see windows 1 and 3 as a tabbed group, with the tabs at the bottom. Sheets 4, 5, and 7 appear as tabbed documents together, as do 6 and 8, both groups with tabs at the top. Window 2 is alone and untabbed.

4.3 Persist the MDI State

You can set the application so that, when the user launches the application, the sheets are open in the same position and state as when the user closed it.

To persist the states of opened sheets, it is important to associate a meaningful string ID with each opened sheet. There are two ways to do this:

- Set it as an argument when using the `OpenSheetAsDocument`, `OpenSheetWithParmAsDocument`, `OpenSheetDocked`, `OpenSheetWithParmDocked`, `OpenSheetInTabGroup`, or `OpenSheetWithParmInTabGroup` function.
- Set it using the `SetSheetID` function.

You can use a function to store the MDI state in the registry when the application closes. You can then use other functions to load and open the sheets in their docking states, and present them.

4.3.1 SetSheetID PowerScript Function

Sets the unique identifier for an open sheet.

Applies to

Window objects

Syntax

```
<controlname>.SetSheetID ( <sheetname> )
```

Argument	Description
<controlname>	The open sheet to be identified.
<sheetname>	A unique string identifier for the sheet, which is used when layout is persisted.

Returns

Integer. Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is null, `SetSheetID` returns null.

Usage

If no sheet indentifier was set when it was opened by one of the OpenSheet functions, you can set an ID using the `SetSheetID` function. You can also change a sheet's ID.

Example

```
window win[]
OpenSheetDocked(win[1], this, WindowDockLeft!, "")
win[1].SetSheetID("sheet1")
```

4.3.2 SaveDockingState PowerScript function

Stores the MDI state in the registry.

Applies to

Window objects

Syntax

```
SaveDockingState ( <regkey> )
```

Argument	Description
<regkey>	The **<regkey>** argument is the registry key. If no entry for the key exists in the registry, one is created. Existing keys are overwritten.

Returns

Integer. Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is null, `SaveDockingState` returns null.

Usage

You can set the application so that it saves the states of the open sheets. You should call this function when the application closes.

Example

Save all sheets in register

```
integer li_rtn
string is_register = "Sybase\PowerBuilder\Examples\Docking\"
li_rtn = this.SaveDockingState (is_register)
```

4.3.3 LoadDockingState PowerScript function

Loads two arrays of equal size: type names of persisted sheets and the corresponding IDs.

Applies to

Window objects

Syntax

```
LoadDockingState ( <regkey>, <windowtypes>, <sheetnames> )
```

Argument	Description
<regkey>	The registry key where the information was stored using the `SaveDockingState` function.
<windowtypes>	A string array of window types for all the child windows that were persisted.
<sheetnames>	A string array of the unique IDs for the persisted child windows corresponding to the types in the **<windowtypes>** array.

Returns

Integer. Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is null, `LoadDockingState` returns null.

Usage

To retrieve the MDI states that were saved using the `SaveDockingState` function, use `LoadDockingState` to get the window information from the registry. Next, use `OpenSheetFromDockingState` or `OpenSheetWithParmFromDockingState` to open each of the persisted sheets. Finally, use `CommitDocking` to do the final arrangement and make the sheets visible.

4.3.4 OpenSheetFromDockingState PowerScript function

Opens one or more persisted sheets within an MDI frame window for dockable windows.

Applies to

Windows objects

Syntax

```
OpenSheetFromDockingState ( <sheetrefvar> {, <windowtype> }, <mdiframe>,
<sheetname> )
```

Argument	Description
<sheetrefvar>	The name of any window variable that is not an MDI frame window. `OpenSheetFromDockingState` places a reference to the open sheet in **<sheetrefvar>**.
<windowtype> (optional)	A string whose value is the datatype of the window you want to open. The datatype of **<windowtype>** must be the same or a descendant of **<sheetrefvar>**.
<mdiframe>	The name of an MDI frame window.
<sheetname>	The unique string identifier for the sheet.

Returns

Integer. Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is null, `OpenSheetFromDockingState` returns null.

Usage

Open persisted sheets in their saved docking states.

4.3.5 OpenSheetWithParmFromDockingState PowerScript function

Opens one or more persisted sheets within an MDI frame window for dockable windows. `OpenSheetWithParmFromDockingState` also stores a parameter in the system's Message object so that it is accessible to the opened sheet.

Applies to

Windows objects

Syntax

```
OpenSheetWithParmFromDockingState ( <sheetrefvar>, <parameter> {,
<windowtype> }, <mdiframe>, <sheetname> )
```

Argument	Description
<sheetrefvar>	The name of any window variable that is not an MDI frame window. `OpenSheetWithParmFromDockingState` places a reference to the open sheet in **<sheetrefvar>**.
<parameter>	The parameter you want to store in the Message object when the sheet is opened. **<Parameter>** must have one of these datatypes: • String • Double • PowerObject
<windowtype> (optional)	A string whose value is the datatype of the window you want to open. The datatype of **<windowtype>** must be the same or a descendant of **<sheetrefvar>**.
<mdiframe>	The name of an MDI frame window.
<sheetname>	The unique string identifier for the sheet.

Returns

Integer. Returns 1 if it succeeds and -1 if an error occurs. If any argument's value is null, `OpenSheetWithParmFromDockingState` returns null.

Usage

The system Message object has three properties for storing data. Depending on the datatype of the parameter specified for `OpenSheetWithParmFromDockingState`, scripts for the opened sheet would check one of the following properties.

Message object property	Argument datatype
Message.DoubleParm	Double
Message.PowerObjectParm	PowerObject (PowerBuilder objects, including user-defined structures)
Message.StringParm	String

In the opened window, it is a good idea to access the value passed in the Message object immediately (because some other script may use the Message object for another purpose).

> **Note**
>
> When you pass a PowerObject as a parameter, you are passing a reference to the object. The object must exist when you refer to it later or you get a null object reference, which causes an error. For example, if you pass the name of a control on a window that is being closed, that control will not exist when a script accesses the parameter.

4.3.6 CommitDocking PowerScript function

After all persisted sheets are opened, this function arranges them and makes them visible.

Applies To

Window objects

Syntax

```
CommitDocking()
```

Usage

When all persisted sheets are opened using the `LoadDockingState` and `OpenSheetFromDockingState` or `OpenSheetWithParmFromDockingState`, the `CommitDocking` function does the work of arranging everything in place and making it all visible.

Example

Restore all sheets for register

```
string s1[], s2[]
string is_register = "Sybase\PowerBuilder\Examples\Docking\"
integer li_start, li_end, li_i, li_rtn
li_rtn = LoadDockingState(is_register,s1,s2)
window lw_window
li_start = lowerbound(s1)
li_end = upperbound(s2)

for li_i = li_start to li_end
    openSheetFromDockingState(lw_window,s1[li_i], this, s2[li_i])
next
CommitDocking()
```

4.4 Properties for Dockable Windows

Customize the behavior and appearance of the windows and tabs.

These properties are similar to existing properties for

Options for Child Windows

WindowDockOptions are for child windows to specify how they can be opened:

- WindowDockOptionAll!
- WindowDockOptionTabbedDocumentOnly!
- WindowDockOptionDockedOnly!
- WindowDockOptionFloatOnly!
- WindowDockOptionTabbedDocumentAndDockedOnly!
- WindowDockOptionTabbedDocumentAndFloatOnly!
- WindowDockOptionDockedAndFloatOnly!

Tabs

Properties for the shapes of the tabs:

- windowdocktabslanted!
- windowdocktabrectangular!
- windowdocktabsingleslanted!

Location of the close button on a tab, if any:

- windowdocktabclosebuttonnone!
- windowdocktabclosebuttononactive!
- windowdocktabclosebuttonshared!

Colors of tabs:

- gradients are available
- default to theme colors is available

Icon and Scroll Button for tabbed windows or documents:

- TabbedWindowTabIcon / TabbedDocumentTabIcon
- TabbedWindowTabScroll / TabbedDocumentTabScroll

Tabbed Window and Document Title Bars

Colors of tabbed window and document title bars:

- TabbedWindowActiveTabBackColor / TabbedDocumentActiveTabBackColor
- TabbedWindowActiveTabGradientBackColor / TabbedDocumentActiveTabGradientBackColor
- TabbedWindowActiveTabTextColor / TabbedDocumentActiveTabTextColor
- TabbedWindowInActiveTabBackColor / TabbedDocumentInActiveTabBackColor
- TabbedWindowInActiveTabGradientBackColor / TabbedDocumentInActiveTabGradientBackColor
- TabbedWindowInActiveTabTextColor / TabbedDocumentInActiveTabTextColor
- TabbedWindowMouseoverTabBackColor / TabbedDocumentMouseoverTabBackColor
- TabbedWindowMouseoverTabGradientBackColor / TabbedDocumentMouseoverTabGradientBackColor
- TabbedWindowMouseoverTabTextColor / TabbedDocumentMouseoverTabTextColor
- TabbedWindowTabsAreaColor / TabbedDocumentTabsAreaColor
- TabbedWindowTabsAreaGradientColor / TabbedDocumentTabsAreaGradientColor
- TabbedWindowTabsAreaGradientVert / TabbedDocumentTabsAreaGradientVert

Title bar states:

- TitleBarActiveColor / TitleBarInActiveColor
- TitleBarActiveGradientColor / TitleBarInActiveGradientColor
- TitleBarActiveGradientVert / TitleBarInActiveGradientVert
- TitleBarActiveaTextColor / TitleBarInActiveTextColor

Important Disclaimers on Legal Aspects

This document is for informational purposes only. Its content is subject to change without notice, and SAP does not warrant that it is error-free. SAP MAKES NO WARRANTIES, EXPRESS OR IMPLIED, OR OF MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE.

Coding Samples

Any software coding and/or code lines / strings ("Code") included in this documentation are only examples and are not intended to be used in a productive system environment. The Code is only intended to better explain and visualize the syntax and phrasing rules of certain coding. SAP does not warrant the correctness and completeness of the Code given herein, and SAP shall not be liable for errors or damages caused by the usage of the Code, unless damages were caused by SAP intentionally or by SAP's gross negligence.

Accessibility

The information contained in the SAP documentation represents SAP's current view of accessibility criteria as of the date of publication; it is in no way intended to be a binding guideline on how to ensure accessibility of software products. SAP specifically disclaims any liability with respect to this document and no contractual obligations or commitments are formed either directly or indirectly by this document.

Gender-Neutral Language

As far as possible, SAP documentation is gender neutral. Depending on the context, the reader is addressed directly with "you", or a gender-neutral noun (such as "sales person" or "working days") is used. If when referring to members of both sexes, however, the third-person singular cannot be avoided or a gender-neutral noun does not exist, SAP reserves the right to use the masculine form of the noun and pronoun. This is to ensure that the documentation remains comprehensible.

Internet Hyperlinks

The SAP documentation may contain hyperlinks to the Internet. These hyperlinks are intended to serve as a hint about where to find related information. SAP does not warrant the availability and correctness of this related information or the ability of this information to serve a particular purpose. SAP shall not be liable for any damages caused by the use of related information unless damages have been caused by SAP's gross negligence or willful misconduct. Regarding link classification, see: http://help.sap.com/disclaimer.

www.sap.com/contactsap