Application Library

VERSION 4.0

PowerBuilder

December 1994

Contents

About This Manual

Subject

This manual describes how to use the PowerBuilder Application Library, a collection of reusable objects that you can use to accelerate the development process. The manual includes installation instructions, a tutorial, an object reference, and usage instructions for the sample applications included with the Application Library.

Audience

This manual is for anyone who will be building applications with PowerBuilder. It assumes that:

- You are familiar with the user interface guidelines for the computing platform you will be developing and deploying your applications on. If not, consult a book that covers the user-interface conventions.
- You are currently developing applications using PowerBuilder and understand the concepts and techniques described in the PowerBuilder *Building Applications* manual.
- You understand SQL and how to use your site-specific DBMS.

PART ONE

Getting Started

This part describes the Application Library and how to install it.

CHAPTER 1

Application Library Overview

About this chapter

This chapter introduces the Application Library and its components. It also describes application development, sample applications, and code examples.

Contents

Introduction to the Application Library

The PowerBuilder Application Library is a collection of PowerBuilder objects that speed the PowerBuilder application development process.

Two sets of objects

The Application Library contains two sets of objects:

- The **application framework** provides a set of ancestor objects for developing a PowerBuilder multiple document interface (MDI) application. You create the core of your application by inheriting from application framework objects.
- The **object library** provides reusable objects that you incorporate into your application.

Application framework = ancestor objects

An application framework provides ancestor objects (base classes) from which objects in your application can inherit. The SYS.PBL library contains the Application Library's application framework objects.

The ancestor objects

The application framework's ancestor objects include windows, a menu, a function, and user objects.

Ancestor object	Description	Comment
w_sys_frame	Displays the application frame window	
w_sys_single_dw	Displays a single-row DataWindow	
w_sys_multi_dw	Displays a multi-row DataWindow	Descendant of w_sys_single_dw
w_sys_shared_dw	Displays two DataWindows in a Master/Detail relationship (uses the shared DataWindow technique)	Descendant of w_sys_multi_dw
w_sys_mast_detl_dw	Displays two DataWindows in a many-to-many relationship	
w_sys_pipeline	Contains data pipeline functionality	
w_sys_report	Displays reports	
m_sys_frame	Provides the base menu for all applications	

Ancestor object	Description	Comment
uo_dw	Provides a DataWindow control used in application framework windows	
u_ole	Provides an OLE 2.0 window control and basic functions for using OLE	
u_ole_excel	Provides an OLE 2.0 window control with functions for use with Microsoft Excel	Descendant of u_ole
u_ole_word	Provides an OLE 2.0 window control with functions for use with Microsoft Word for Windows	Descendant of u_ole
u_pipeline_kit	Provides functions for use with data pipelines	
f_app_open	Initializes global variables	

When you create an object inherited from an application framework object, you get all of its characteristics and behaviors in your object. The ancestor object's characteristics and behaviors include encapsulated events, user events, and functions, which you can use as-is, extend, override, trigger, post, or call, depending on your application's needs.

For more on application framework objects and functions, see Part Three "Object Reference." For more on using an application framework, see *Building Applications* in the PowerBuilder documentation set.

Object library = reusable objects

The Application Library's object library provides reusable objects that your application can use. These objects, which perform error handling and other utility functions, can enhance your application's reliability and functionality. The UTLFUNC.PBL and UTLWIN.PBL libraries contain the object library.

The application framework uses the object library
The application framework makes extensive use of object library functions, windows, and structures.

Six types of reusable objects

The object library includes windows, functions, a menu, user objects, structures, and DataWindows.

Object type	Description	Comment
Windows	Windows that you can use for a particular purpose, such as sorting or displaying error information. You open some windows using Application Library function calls and others using the Open function.	Some windows are part of the application framework, which means that you use them as ancestor objects. All other windows you can use as-is
DataWindows	DataWindow objects that support other object library windows and functions.	These objects do not perform any database access. Not typically used in your application.
Functions	Global functions that perform specific processing.	Processing can include return values, window display, and manipulation of nonvisual attributes. UTLFUNC.PBL contains standalone functions; UTLWIN.PBL contains global functions that display a window.
Structures	Collections of related variables that the Application Library uses for communications purposes.	Most of these are intended for use with other objects; others provide general functionality.
Menu	A general-purpose menu that contains basic File and Window menu items.	Use this as an ancestor object.
User objects	User objects with encapsulated functionality.	These objects allow you to define a component once and then reuse it as many times as you need without any additional work.

Creating an application

❖ To create an application using the Application Library:

1 Create an application object.

2 Add the SYS.PBL, UTLFUNC.PBL, and UTLWIN.PBL libraries to your application's library search path.

Failure to do this will cause errors.

3 Create an MDI frame window by inheriting from the w_sys_frame window.

> **Remember**
> The application framework is designed to create MDI applications.

4 Create MDI sheet windows by inheriting from the appropriate application framework sheet window:

- w_sys_mast_detl_dw
- w_sys_multi_dw
- w_sys_pipeline
- w_sys_report
- w_sys_shared_dw
- w_sys_single_dw

5 Create a frame menu by inheriting from the m_sys_frame menu.

6 Add or modify frame menu items to meet your application's needs.

7 Create sheet menus by inheriting from your frame menu.

8 Modify sheet menus to meet the sheet's needs.

9 Associate sheet menus with corresponding sheet windows.

10 Add application-specific logic to the application.

Object library objects can help you implement application-specific logic.

Getting started with the Application Library

The Application Library contains many objects. Take the time to learn as much as possible about the Application Library before creating your first production application. The Application Library provides these resources to help you get started:

- **This manual** The *Application Library* manual includes overview, tutorial, and reference information.
- **Tutorial** The tutorial provided in Part Two steps through the creation of a small application and demonstrates the usage of application framework and object library objects.
- **Sample applications** The Application Library includes two sample applications made up of Application Library objects. By running these applications and examining their windows and events, you can observe the structure, look, and feel of applications created using the Application Library.
- **Code examples** Also included in the Application Library is a frontend application demonstrating the usage of Application Library objects that are not used in the sample applications. These code examples allow you to execute a series of windows that show working examples of Application Library functions and objects.

CHAPTER 2

Installation

About this chapter

This chapter describes how to install the Application Library and its supporting files.

Contents

Installing the PowerBuilder Application Library

The PowerBuilder Application Library Setup program installs the Application Library in the directory on the drive you specify or in the default directory, c:\pbapp.

❖ To install the Application Library:

1 Run the PowerBuilder Application Library Setup program:

- To run from the DOS prompt, type:

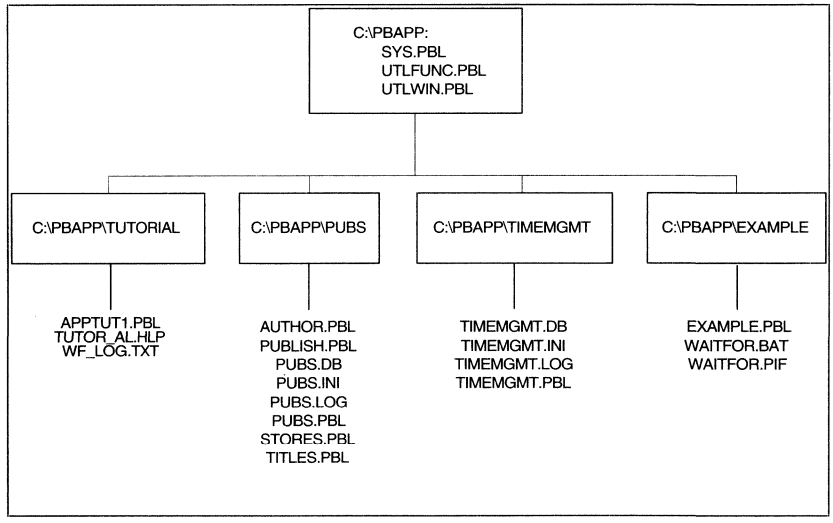
  WIN A:\SETUP

- To run from the Windows Program Manager, select File➤Run from the menu bar and type:

  A:SETUP

2 Respond to the Setup prompts and select the options you want.

The Setup program creates a default directory c:\pbapp for the PowerBuilder Application Library files. The c:\pbapp directory contains the main Application Library components and subdirectories for the tutorial, sample applications, and code examples.



For installation information about sample applications and code examples, see Chapters 10, 11, and 12.

Verifying the installation

❖ **Verify that the following files are in the following directories:**

Directory	Files
C:\PBAPP	SYS.PBL UTLFUNC.PBL UTLWIN.PBL
C:\PBAPP\EXAMPLE	EXAMPLE.PBL WAITFOR.BAT WAITFOR.PIF
C:\PBAPP\PUBS	AUTHOR.PBL PUBLISH.PBL PUBS.DB PUBS.INI PUBS.LOG PUBS.PBL STORES.PBL TITLES.PBL
C:\PBAPP\TIMEMGMT	TIMEMGMT.DB TIMEMGMT.INI TIMEMGMT.LOG TIMEMGMT.PBL
C:\PBAPP\TUTORIAL	APPTUT1.PBL TUTOR_AL.HLP WF_LOG.TXT

PART TWO

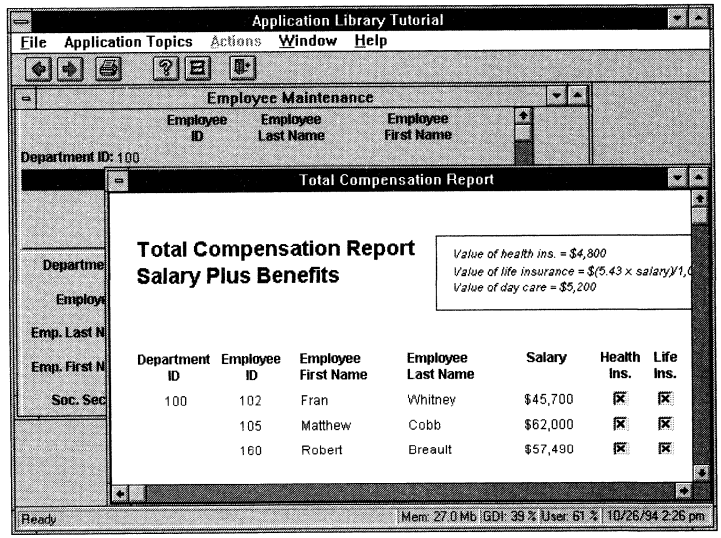
TUTORIAL

This part provides a series of eight lessons in which you build a simple application using Application Library objects.

CHAPTER 3

Setting Up for the Tutorial

The Application Library tutorial is a series of eight lessons in which you build a small MDI application using Application Library objects. The finished application looks like this.



The sheet windows that you build in the Application Library tutorial application are:

- **A basic Master/Detail window** The top half of the window contains a list of employees with a pointer to a single employee; the bottom half of the window displays extra detail for the current employee. The DataWindows are linked by the shared DataWindow feature, which you get automatically by inheriting from the w_sys_shared_dw window.
- **A report window** The window contains an employee compensation report in a single DataWindow. It is designed for online viewing and includes zoom in, zoom out, and print preview, which you get automatically by inheriting from the w_sys_report window.

For more on frame windows, sheet windows, and MDI applications, see *Building Applications* in the PowerBuilder documentation set.

This application is similar to the PowerBuilder tutorial

The application you create in this tutorial is similar to the one you created in the PowerBuilder *Getting Started* tutorial.

If you have already completed the PowerBuilder *Getting Started* tutorial, compare that application with this one. You will notice many places where the Application Library provides enhanced error checking, ease of use, and flexibility.

If you are new to PowerBuilder and have not yet completed the *Getting Started* tutorial, consider reviewing that tutorial before using this tutorial.

What you will do

Lesson 1	You will begin by using the PowerBuilder Application painter to create the Application object, update the library search path, create scripts, and associate an icon with the application.
Lessons 2 through 4	Then you will use the Window painter to create the frame window and the sheet windows for the application by inheriting from Application Library windows. You will also add scripts for the application, the frame window, and the sheet windows.
Lessons 5 through 7	Next you will create menus for the application.
Lesson 8	Finally you will associate DataWindow objects with the DataWindow controls in the sheet windows and test the application.

How long it will take

You can do the entire tutorial in one sitting in about four to five hours. Or you can stop after any lesson and continue at another time.

What you will learn

You will learn basic Application Library techniques and concepts, including how to:

- **Use the Application painter** to include Application Library libraries in the search path
- **Use the Application Library f_app_open function** and application INI file to perform application setup and database connection automatically
- **Use the Window painter** to create frame and sheet windows that are descendants of Application Library application framework windows
- **Use the Menu painter** to create frame and sheet menus that are descendants of the Application Library application framework menu
- **Use the functions and user events** encapsulated in Application Library application framework objects

- **Use Application Library** functions and windows

For more on PowerBuilder application development, see *Building Applications* in the PowerBuilder documentation set.

Assumptions

This tutorial assumes that you have:

- Installed the Watcom SQL DBMS
- Installed and configured the Powersoft Demo DB database

> **Use the PowerBuilder installation diskettes**
> If your system does not contain Watcom SQL or the Powersoft Demo DB, use the PowerBuilder installation diskettes to install them.

LESSON 1

Creating the Application Object

The first step in building a PowerBuilder application is to create an application object. You are always developing within the scope of an application.

In this lesson you will create an application object for an Employee Maintenance application. You will also add the Application Library libraries to the search path, code SystemError and Close event scripts, and specify an icon for the application.

How long will this lesson take?
About 15 minutes.

What will you learn about the Application Library?

- How to use the w_system_error window to trap system errors.
- How to use the f_db_error function to check for database errors in embedded SQL statements.
- How to include the SYS.PBL, UTLFUNC.PBL, and UTLWIN.PBL libraries in the application library search path. These libraries contain all Application Library objects.

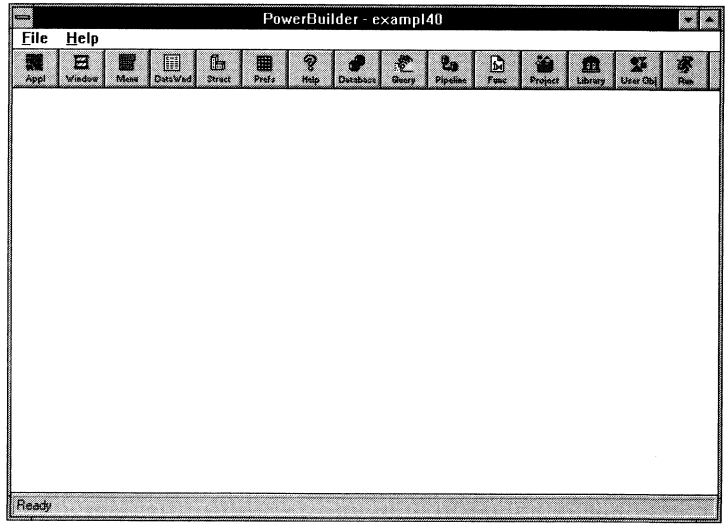
Create and save the application object

Where you are
Lesson 1 Creating the Application Object
➡ Create and save the application object
Update the application library search path
Add a SystemError event script
Add a Close event script
Specify an icon for the application

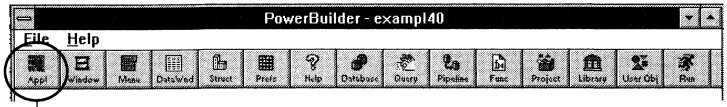
Now you will start PowerBuilder, open the Application painter, create the tutor_al application object, and select the library to hold that object and all the other objects you create in the tutorial.

1 Start PowerBuilder.

The initial window displays.

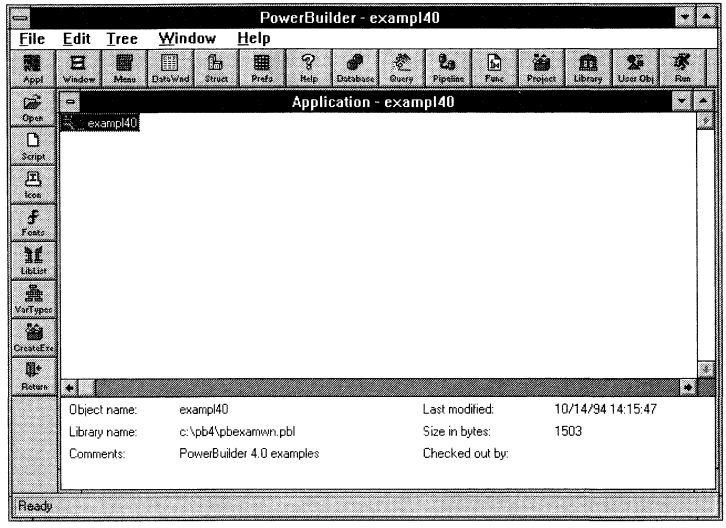


2 Click the *Application painter* button in the PowerBar.



Application painter

The Application painter workspace displays.

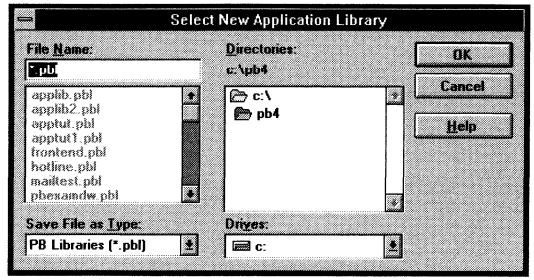


This tutorial uses buttons that show text

The sample windows in this tutorial display buttons that show text. If your PowerBuilder buttons have no text displayed, you can either use **PowerTips** (place the pointer over a button for a few seconds and a brief description appears) or enable button text (right-click in the PowerBar and select Show Text).

3 Select File➤New from the menu bar.

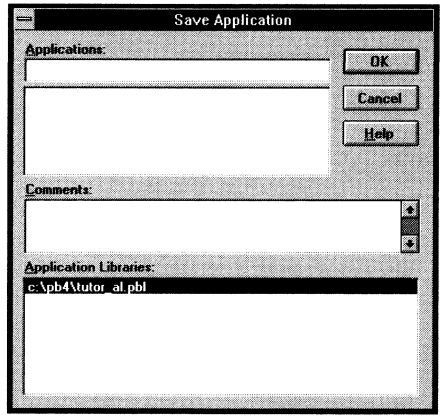
The Select New Application Library dialog box displays.



4 Type *tutor_al.pbl* in the File Name box.

5 Click *OK*.

The Save Application dialog box displays.

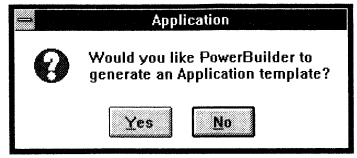


6 Type *tutor_al* in the Applications box.
Press the TAB key twice to move to the Comments box.
Type *This is the Application Library 4.0 tutorial application.*

This associates a descriptive comment with the application object. The comment you add here displays in the Library painter and helps to identify objects. Comments are optional.

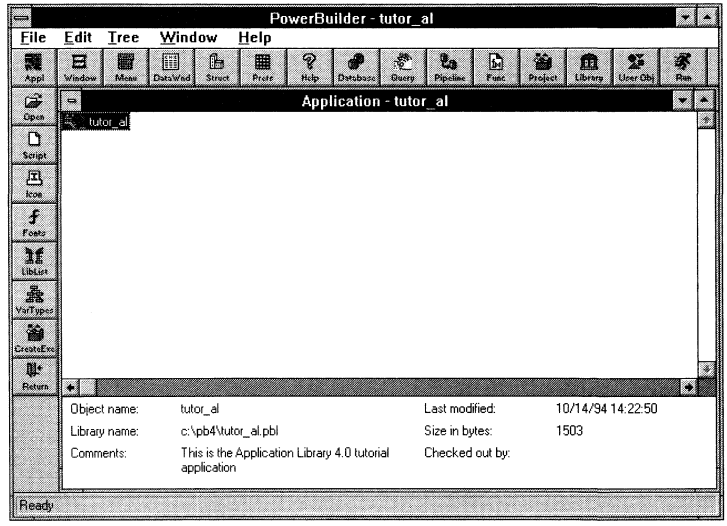
7 Click *OK.*

The Application dialog box displays.



8 Click *No.*

PowerBuilder creates a library named tutor_al.pbl and creates and saves the application object named tutor_al in the library. The Application painter workspace displays.



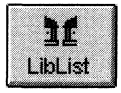
At this point, the tutor_al application is your current application.

Update the application library search path

Where you are
Lesson 1 Creating the Application Object
Create and save the application object
➡ Update the application library search path
Add a SystemError event script
Add a Close event script
Specify an icon for the application

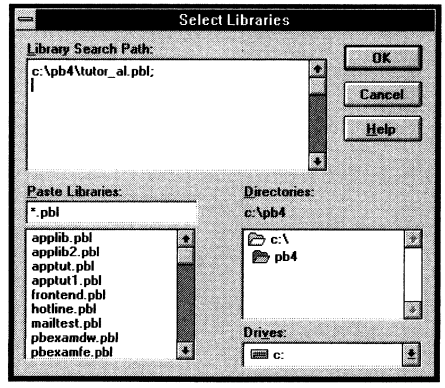
Now you will add these PowerBuilder libraries to the library search path:

- SYS.PBL
- UTLFUNC.PBL
- UTLWIN.PBL
- APPTUT1.PBL

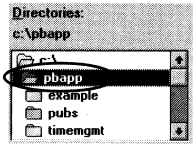


1 Make sure you are in the Application painter. Click the *Library List* button in the PainterBar.

The Select Libraries dialog box displays.



2 Use the *Directories* listbox to access the PBAPP directory.

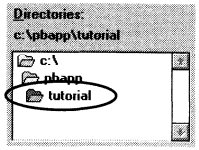


3 **Double-click** *SYS.PBL.*
Double-click *UTLFUNC.PBL.*
Double-click *UTLWIN.PBL.*

PowerBuilder adds the libraries to the search path.

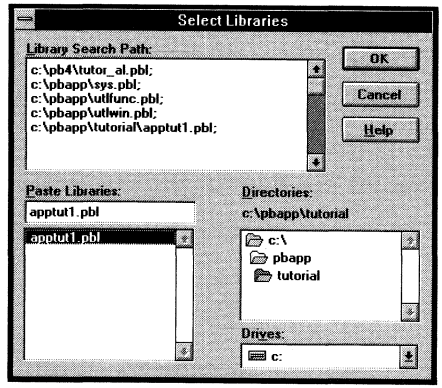
> **This is the recommended order**
> SYS.PBL, UTLFUNC.PBL, and UTLWIN.PBL are the libraries that contain the Application Library objects. The library order used here is the recommended search path when using the Application Library.

4 **Use the Directories listbox to access the PBAPP\TUTORIAL directory.**



5 **Double-click** *APPTUT1.PBL.*

PowerBuilder adds APPTUT1.PBL to the search path. This library contains the DataWindow objects used by the tutorial application.



6 **Click** *OK.*

The Application painter workspace displays.

Add a SystemError event script

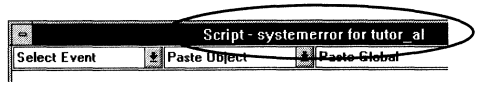
Where you are
Lesson 1 Creating the Application Object
Create and save the application object
Update the application library search path
➡ Add a SystemError event script
Add a Close event script
Specify an icon for the application

Now you will add a script that opens the w_system_error window to the application's SystemError event. If a system error occurs, this window opens, displays system error information, and allows the user to exit the application, continue the application, or print the error message.

1 Click the *Script* button in the PainterBar.

The PowerScript painter displays.

2 Make sure the title reads: *Script - systemerror for tutor_al.*



If the title is incorrect
If the event is not the SystemError event, pull down the Select Event listbox and select SystemError.

3 Type the following script:

```
open(w_system_error)
```

4 Click the Return button.
or
Select File➤Return from the menu bar.

PowerBuilder compiles your script and returns to the Application painter workspace.

Add a Close event script

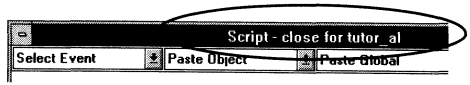
Where you are
Lesson 1 Creating the Application Object
Create and save the application object
Update the application library search path
Add a SystemError event script
➡ Add a Close event script
Specify an icon for the application

Now you will add a script to the application Close event to disconnect from the database. (The application framework automatically connects you to the database but you must execute the disconnect.)

1 Click the *Script* button in the PainterBar.

The PowerScript painter displays.

2 Make sure the title reads: *Script - close for tutor_al.*



If the title is incorrect
If the event is not the Close event, pull down the Select Event listbox and select Close.

3 Type the following script:

```
DISCONNECT using SQLCA;
IF f_db_error(SQLCA,"Disconnect Error") &
   <> 0 THEN
      ROLLBACK using SQLCA;
END IF
```

4 Click the *Return* button.
or
Select File➤Return from the menu bar.

PowerBuilder compiles your script and returns to the Application painter workspace.

Specify an icon for the application

Where you are
Lesson 1 Creating the Application Object
Create and save the application object
Update the application library search path
Add a SystemError event script
Add a Close event script
➡ Specify an icon for the application

Now you will associate your application with an icon. The icon you specify displays in the Windows workspace when you minimize your application while executing it.

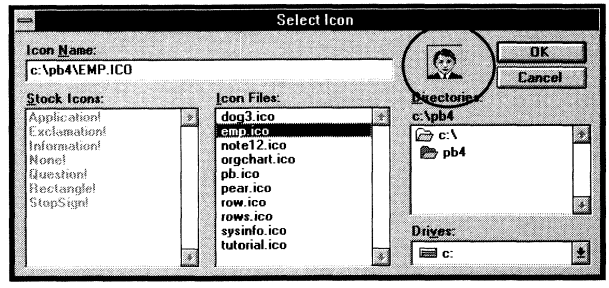
PowerBuilder automatically includes this icon in the executable when you create an EXE file.

1 Click the *Icon* button in the PainterBar.

The Select Icon dialog box lists all available icons.

2 Select *emp.ico*, which is one of the icons delivered with PowerBuilder.

The Employee icon displays in the dialog box.



3 Click *OK*.

The Application painter workspace displays.

4 Select File➤Save from the menu bar.

This saves your application object.

5 **Click the *Return* button on the PainterBar.**

You return to the initial window.

LESSON 2

Building the Frame Window

In an MDI application, you define a window whose type is MDI frame and open other windows as sheets within the frame. The Application Library's application framework is designed to create MDI applications: you use the w_sys_frame window as the ancestor for the frame window and use all other *w_sys*-prefixed windows as ancestors for sheet windows.

SYS.PBL = application framework
The SYS.PBL library contains all application framework ancestor objects. UTLFUNC.PBL and UTLWIN.PBL provide additional reusable objects, but they are not part of the application framework.

In this lesson you will create a frame window by inheriting from the w_sys_frame window. You will also create an application INI file and a script for the application Open event.

How long will this lesson take?
About 15 minutes.

What will you learn about the Application Library?

- How to create a descendant of the w_sys_frame window. Inheriting from w_sys_frame provides many things, including a status line clock (w_mdi_clock), a login window, and automatic database connection.
- How to create an application INI file. This file provides database information that the frame window uses to connect your application to a database.
- How to use the f_app_open function. Using this function provides your application with database independence (because you name the database in the INI file) and control over whether multiple application instances are allowed.

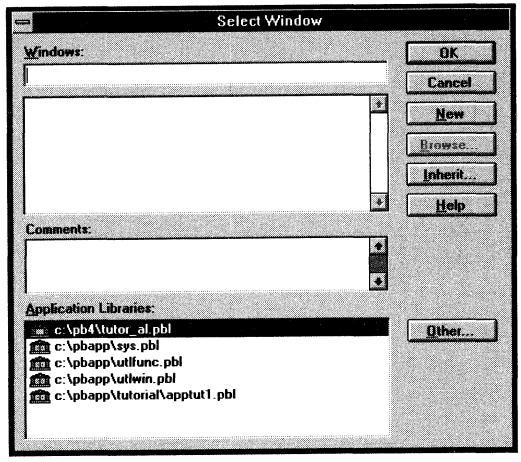
Create and save a descendent window

Where you are
Lesson 2 Building the Frame Window
➡ Create and save a descendent window
Create an application INI file
Add an application script
Run the application

Now you will create a descendent frame window by inheriting from w_sys_frame. Because this is a frame window, you add no window controls.

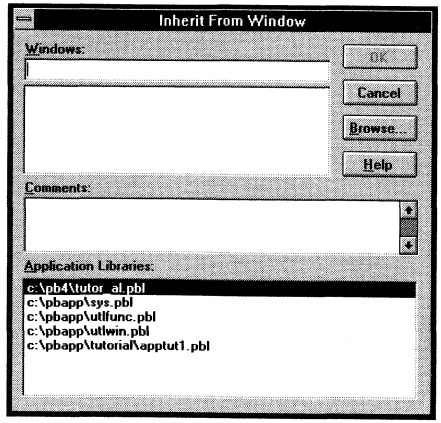
1 Click the *Window painter* button in the PowerBar.

The Select Window dialog box displays.



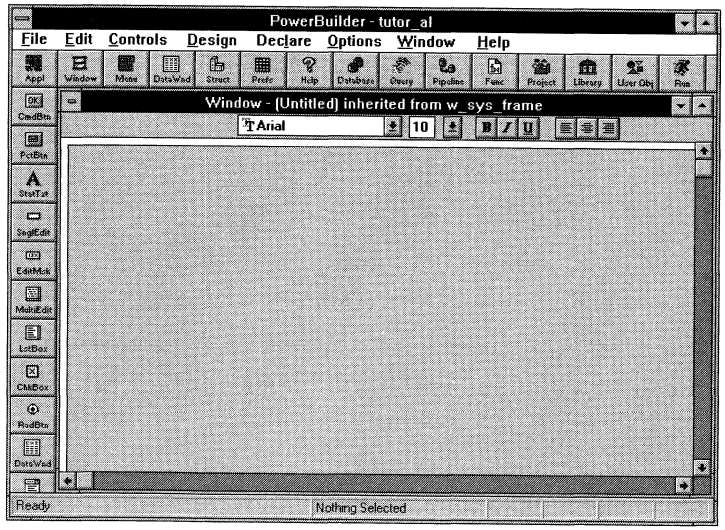
2 Click *Inherit.*

The Inherit From Window dialog box displays.



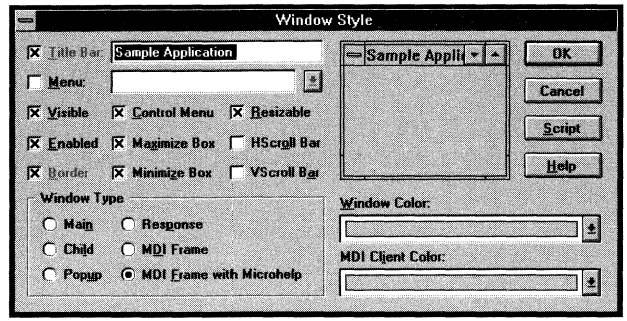
3 Click the line that ends with *sys.pbl* in the Application Libraries box.
Select *w_sys_frame.*
Click *OK.*

The Window painter workspace displays. The title should read: Window - (Untitled) inherited from w_sys_frame.



4 Select Design➤Window Style from the menu bar.

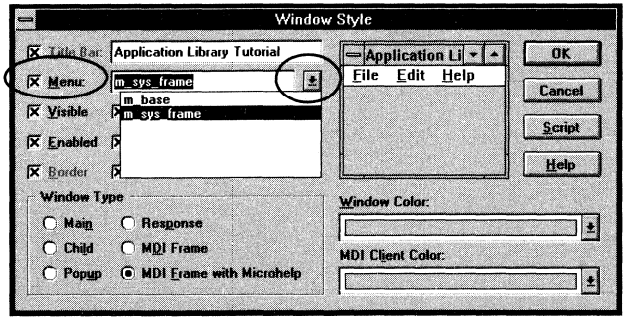
The Window Style dialog box displays. The title bar text is highlighted.



5 Type *Application Library Tutorial*

This replaces Sample Application in the Title Bar box.

6 Select the *Menu* **box.**
Click the down arrow in the dropdown listbox.
Click *m_sys_frame*.



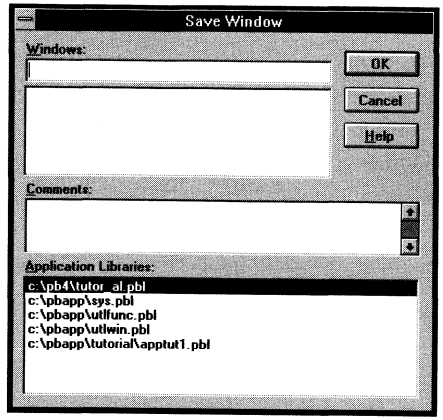
In Lesson 5 you will replace the m_sys_frame specification with a menu customized for the tutor_al application.

7 Click *OK*.

This closes the Window Style dialog box and returns to the Window painter workspace.

8 Select File➢Save As from the menu bar.

The Save Window dialog box displays.



9 Type *w_tut_frame* in the Name box.
Type a comment in the Comments box.
Then click *OK*.

PowerBuilder saves your new window as a descendant of w_sys_frame.

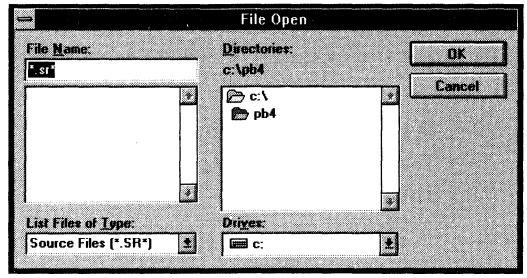
Create an application INI file

Where you are
Lesson 2 Building the Frame Window
Create and save a descendent window
➡ Create an application INI file
Add an application script
Run the application

Now you will create an application INI file. This file provides database information that the f_login function (called in the w_sys_frame window's post_open user event) uses to connect your application to a database. You can also use it to store other default information, which your application can retrieve using the PowerBuilder ProfileString function.

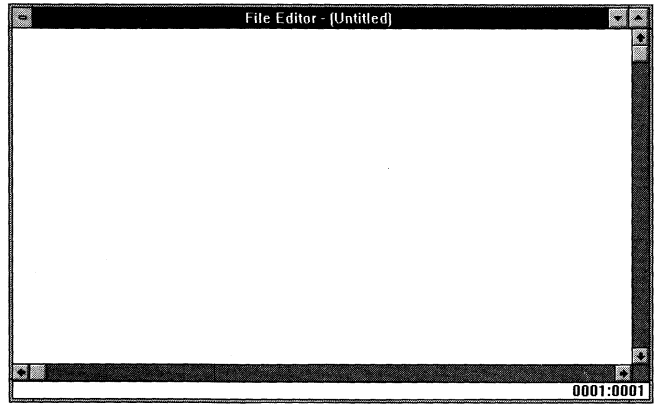
1 Open the PowerBuilder File Editor by pressing SHIFT+F6.

The File Open dialog box displays.

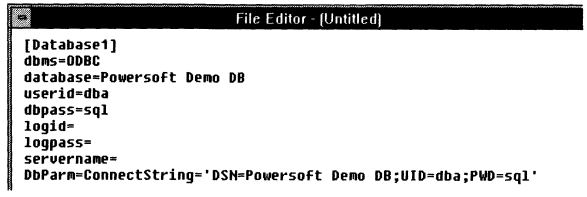


2 Click *Cancel.*

The File Editor displays.



3 Type the following lines.

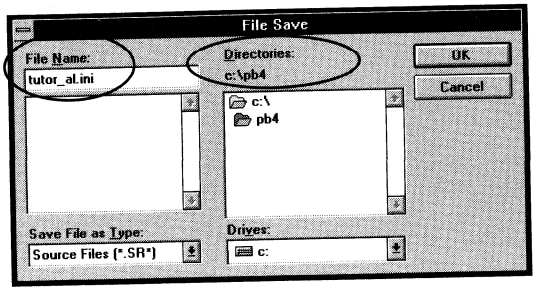


The Database and DSN specifications must match the specification for the Powersoft Demo database in the PB.INI file.

4 Select File➤Save As from the menu bar.

The Save As dialog box displays.

5 **Navigate to the PB4 directory in the Directories box.**
Type *tutor_al.ini* **in the File Name box.**
Click *OK*.



6 **Select File➤Close from the menu bar.**

The File Editor closes.

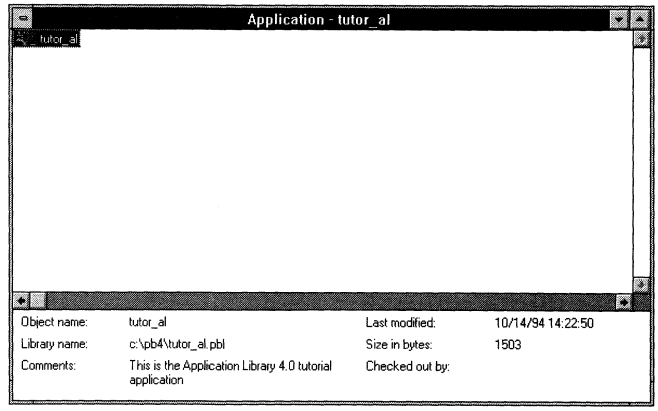
Add an application script

Where you are
Lesson 2 Building the Frame Window
Create and save a descendent window
Create an application INI file
➡ Add an application script
Run the application

Now you code the application Open event. In this event, you call the f_app_open function, which initializes instance variables and opens the frame window.

1 Click the *Application painter* button in the PowerBar.

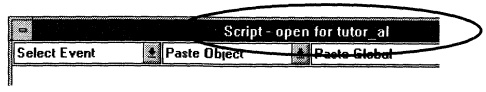
The Application painter displays.



2 Click the *Script* button in the PainterBar.

The PowerScript painter displays.

3 Make sure the title reads: *Script - open for tutor_al.*



If the title is incorrect
If the event is not the Open event, pull down the Select Event listbox and select Open.

4 **Type the following script:**

```
f_app_open("tutor_al.ini",w_tut_frame,FALSE)
```

This script specifies the INI file to be used by the application, the frame window to be opened, and whether multiple instances of the application can be run. The frame window uses this information to control the application.

5 **Click the *Return* button.**
or
Select File➤Return from the menu bar.

PowerBuilder compiles your script and returns to the Application painter workspace.

Run the application

Where you are
Lesson 2 Building the Frame Window
Create and save a descendent window
Create an application INI file
Add an application script
➡ Run the application

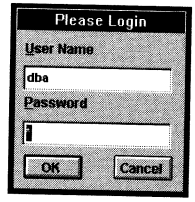
Now you will verify that the database connection is working by running the Application Library tutorial application.

1 **Click the *Run* button in the PowerBar.**
or
Select File➢Run from the menu bar.

PowerBuilder prompts you to save changes.

2 **Click *Yes*.**

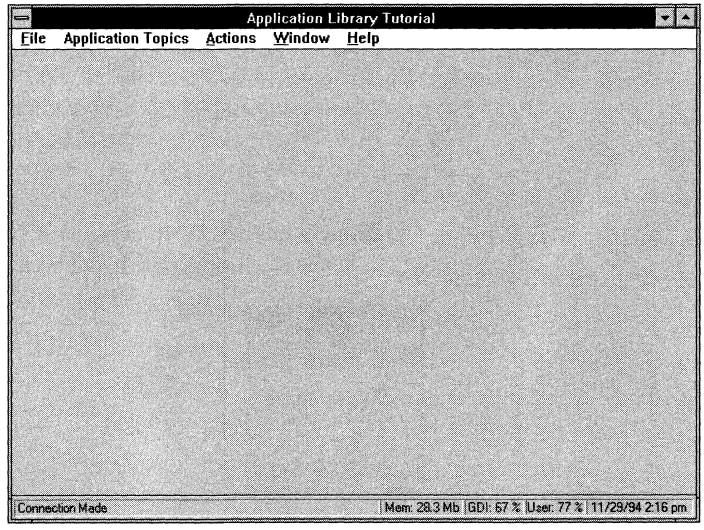
The frame window's Open event script calls the f_login function, which displays the w_login dialog box.



3 Type *sql* (or whatever password you specified in the tutor_al.ini file). Click *OK*.

The application connects to the database and displays the Application Library tutorial frame window.

> **If you cannot connect to the database**
> If your application has trouble connecting to the database, check that the TUTOR_AL.INI file is defined properly. You may need to look at the PB.INI file (in the PB4 directory) to obtain the proper settings for Database and Dbparm. If you are using a database other than Watcom, you may need to fill in other TUTOR_AL.INI parameters, as shown in the PB.INI file.



Notice the information displayed in the status bar at the bottom. This is the w_mdi_clock window, which w_sys_frame (the ancestor of w_tut_frame) automatically displays over the frame window and positions in the lower-right corner.

4 Exit the application by pressing ALT+F4.

The Application painter workspace displays.

LESSON 3

Building the First Sheet Window

You inherit from application framework sheet windows to create the core of your application. Application framework sheet windows include DataWindow controls as well as predefined window functions, events, and user events. These window functions, events, and user events perform basic housekeeping and database access tasks, allowing you to concentrate on your application's processing needs.

In this lesson you will create a sheet window by inheriting from the w_sys_shared_dw window. You will also add a script to the frame window's post_open user event to open the sheet window.

How long will this lesson take?
About 15 minutes.

What will you learn about the Application Library?

- How to create a descendant of the w_sys_shared_dw window. Inheriting from w_sys_shared_dw provides many things, including shared DataWindow setup, a CloseQuery event that prompts users to save changes when exiting the window, and user events that perform database management functions (add, delete, save).
- How to add a Retrieve function to the ue_retrieve_data user event.

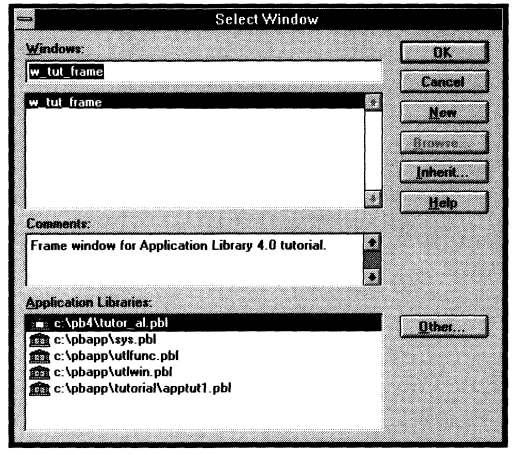
Create a descendent window

Where you are
Lesson 3 Building the First Sheet Window
➡ Create a descendent window
Add a script to the sheet window and save it
Add a script to the frame window
Run the application

Now you will create a sheet window by inheriting from the w_sys_shared_dw window. You will associate this window with a menu in Lesson 6.

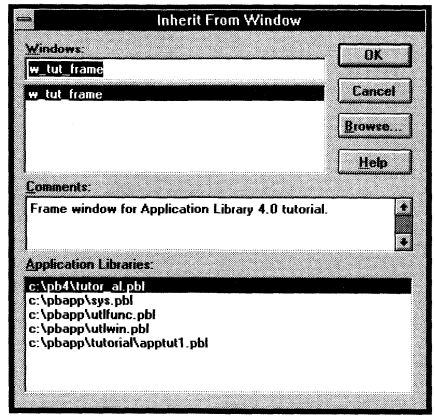
1 Click the *Window painter* button in the PowerBar.

The Select Window dialog box displays.



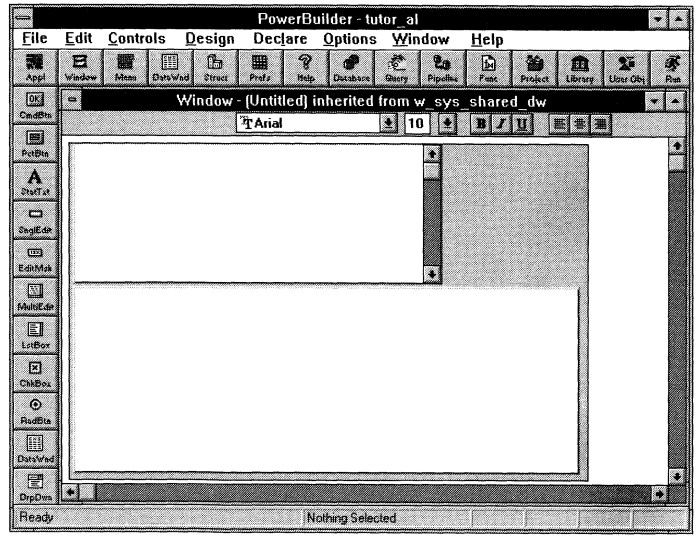
2 Click *Inherit.*

The Inherit From Window dialog box displays.



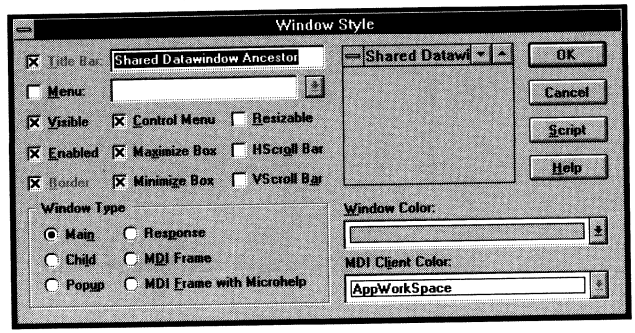
3 Click the line that ends with *sys.pbl* in the Application Libraries box.
Select *w_sys_shared_dw.*
Click *OK.*

The Window painter workspace displays. The title should read: Window - (Untitled) inherited from w_sys_shared_dw.



4 Select Design➤Window Style from the menu bar.

The Window Style dialog box displays. The title bar text is highlighted.



5 Type *Employee Maintenance*

This replaces Shared Datawindow Ancestor in the Title Bar box.

6 Click *OK.*

This closes the Window Style dialog box and returns to the Window painter workspace.

Add a script to the sheet window and save it

Where you are
Lesson 3 Building the First Sheet Window
Create a descendent window
➡ Add a script to the sheet window and save it
Add a script to the frame window
Run the application

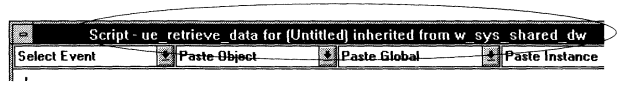
Now you will code a Retrieve function in the ue_retrieve_data user event.

The Application Library includes database error checking
The dw_sheet DataWindow control in the w_sys_shared_dw ancestor includes database error checking (in the DBError event in the uo_dw DataWindow user object) so that you don't have to check for Retrieve errors in your script.

1 Click the *Script* button in the PainterBar.

The PowerScript painter displays.

2 Make sure the title reads: *Script - ue_retrieve_data for (Untitled) Inherited from w_sys_shared_dw.*



If the title is incorrect
If you've accidentally selected one of the DataWindow controls, select Edit➤Select Object from the menu bar and select (Untitled).

If the event is not the ue_retrieve_data event, pull down the Select Event listbox and select ue_retrieve_data.

3 Type the following script:

```
dw_sheet.Retrieve()
```

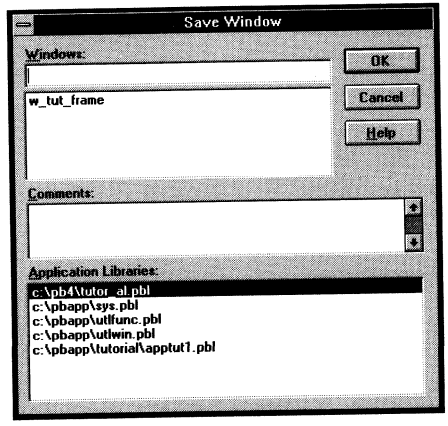
This will retrieve rows, as defined in the DataWindow object.

4 **Click the *Return* button.**
or
Select File➤Return from the menu bar.

PowerBuilder compiles your script and returns to the Window painter workspace.

5 **Select File➤Save As from the menu bar.**

The Save Window dialog box displays.



6 **Type *w_tut_shared* in the Name box.**
Enter a comment in the Comment box.
Click *OK*.

PowerBuilder saves your new window as a descendant of w_sys_shared_dw.

Add a script to the frame window

Where you are
Lesson 3 Building the First Sheet Window
Create a descendent window
Add a script to the sheet window and save it
➡ Add a script to the frame window
Run the application

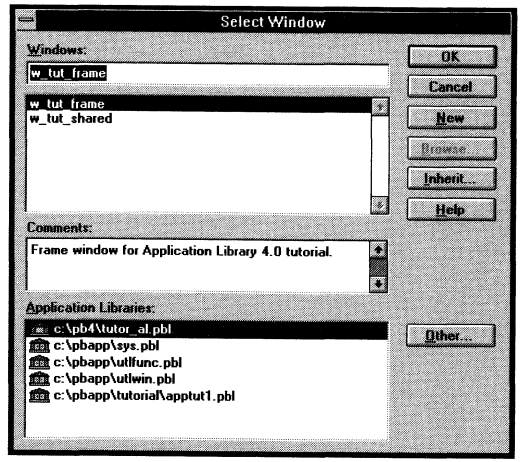
Now you code an OpenSheet function in the w_tut_frame post_open event. This opens the w_tut_shared window when the application is opened. Alternatively you could start with an empty frame window and let the user open the sheet window explicitly using a menu item.

At execution time, the w_tut_frame window's Open event invokes the post_open event using the PostEvent function. Delaying DataWindow retrieval by posting to a user event speeds window display.

Retrieval arguments
If you develop applications whose DataWindow objects use retrieval arguments, you can open the sheet window using the OpenSheetWithParm function, access the arguments in the Message object, and specify the arguments in the Retrieve function.

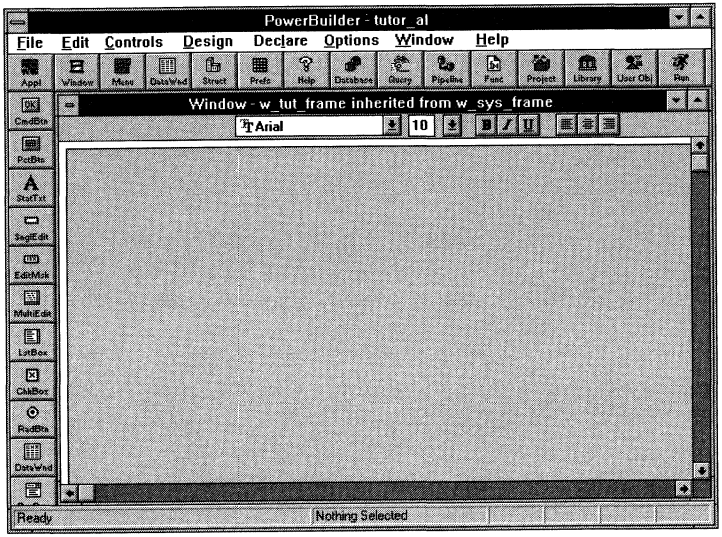
1 **Click the *Window painter* button in the PowerBar.**

The Select Window dialog box displays.



2 **Double-click** *w_tut_frame.*

The Window painter workspace displays.



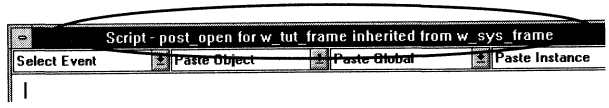
3 **Move the pointer to the window and right-click.**

The popup menu for the window displays.

4 **Click** *Script.*

The PowerScript painter displays.

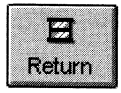
5 **Make sure the title reads:** *Script - post_open for w_tut_frame inherited from w_sys_frame.*



If the title is incorrect
If the event is not the post_open event, pull down the Select Event listbox and select post_open.

6 **Type the following script:**

```
OpenSheet(w_tut_shared,this,0,Cascaded!)
```



7 **Click the *Return* button.**
or
Select File➤Return from the menu bar.

PowerBuilder compiles your script and returns to the Window painter workspace.

8 **Select File➤Save from the menu bar.**

PowerBuilder saves the frame window.

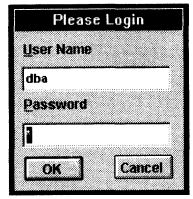
Run the application

Where you are
Lesson 3 Building the First Sheet Window
Create a descendent window
Add a script to the sheet window and save it
Add a script to the frame window
➡ Run the application

Now you will verify that the sheet window opens properly by running the Application Library tutorial application.

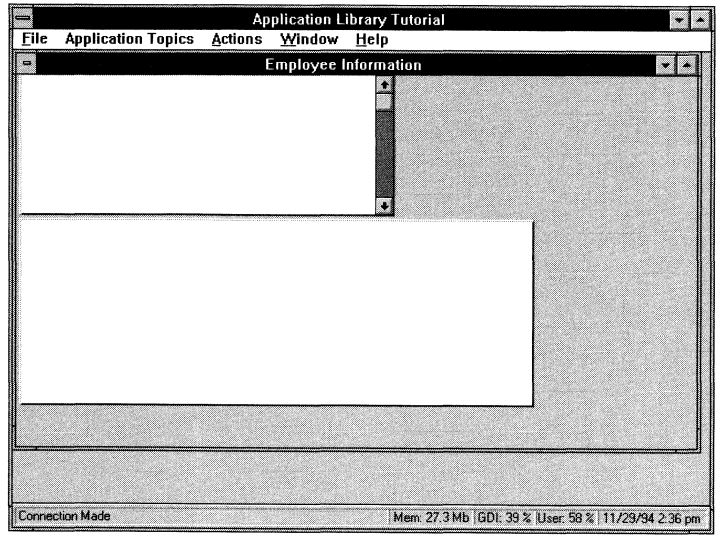
1 **Click the *Run* button in the PowerBar.**
or
Select File➤Run from the menu bar.
(If PowerBuilder prompts you to save changes, click *Yes*.)

The frame window's Open event calls the f_login function, which displays the w_login dialog box.



2 Type *sql* (or whatever password you specified in the tutor_al.ini file). Click *OK*.

The application connects to the database and displays the Application Library Tutorial sheet window.



3 Exit the application by pressing ALT+F4.

The Window painter workspace displays.

LESSON 4

Building the Second Sheet Window

In this lesson you will create a sheet window by inheriting from the w_sys_report window.

How long will this lesson take?
About 15 minutes.

What will you learn about the Application Library?

- How to create a descendant of the w_sys_report window. Inheriting from w_sys_report provides many things, including zoom in, zoom out, print preview, and query mode.

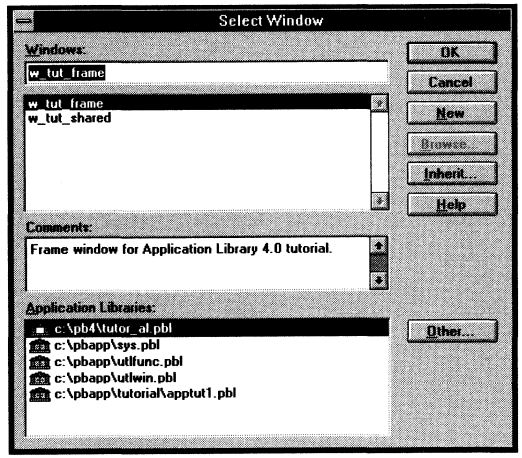
Create a descendent window

Where you are
Lesson 4 Building the Second Sheet Window
➡ Create a descendent window
Add a script to the sheet window and save it

Now you will create a sheet window by inheriting from the w_sys_report window. You will associate this window with a menu in Lesson 7.

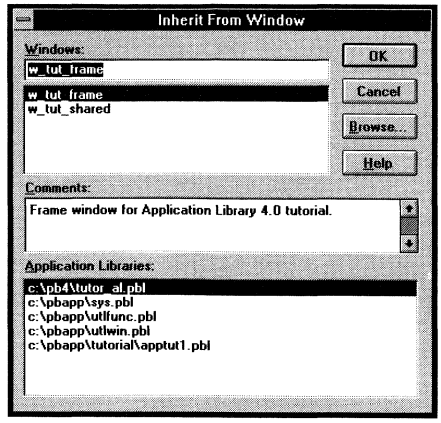
1 Click the *Window painter* button in the PowerBar.

The Select Window dialog box displays.



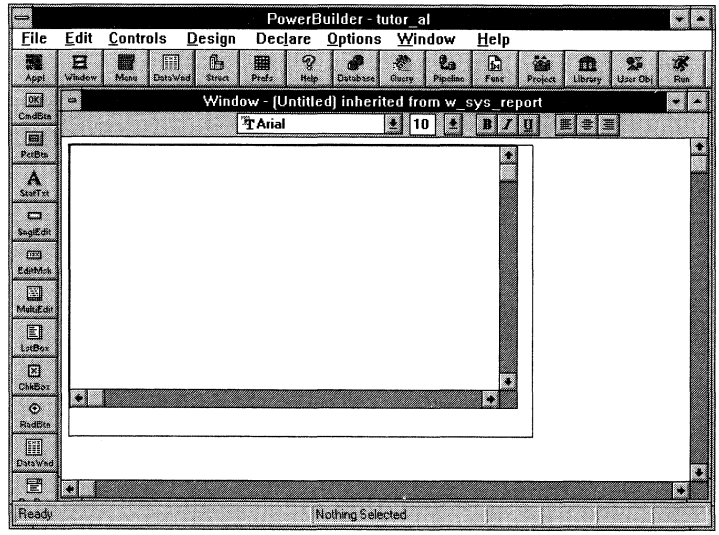
2 Click *Inherit*.

The Inherit From Window dialog box displays.



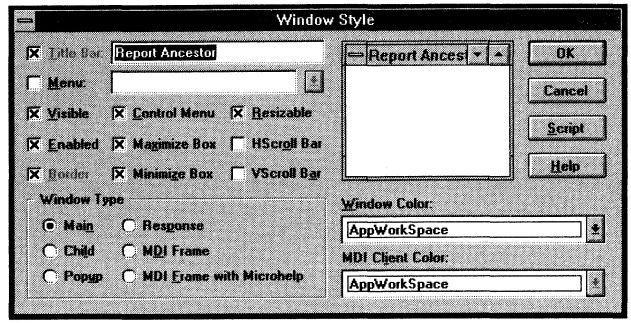
3 Click the line that ends with *sys.pbl* in the Application Libraries box.
Select *w_sys_report*.
Click *OK*.

The Window painter workspace displays. The title should read: Window - (Untitled) inherited from w_sys_report.



4 Select Design➤Window Style from the menu bar.

The Window Style dialog box displays. The title bar text is highlighted.



5 Type *Total Compensation Report*

This replaces Report Ancestor in the Title Bar box.

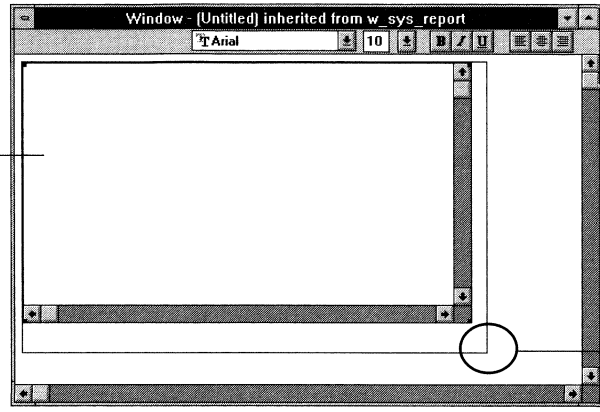
6 Click *OK.*

This closes the Window Style dialog box and returns to the Window painter workspace.

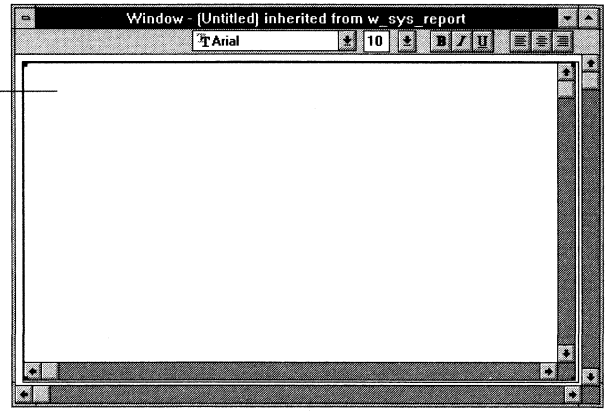
7 Make the window rectangle larger.
Make the DataWindow control larger.

To do this, move the pointer to the lower-right corner of the window rectangle. When it turns into a double-headed arrow, drag the arrow down and to the right to enlarge the rectangle. Make it almost as large as the Window painter workspace. Click in the DataWindow control and do the same thing.

Before:



After:



Add a script to the sheet window and save it

Where you are
Lesson 4 Building the Second Sheet Window
Create a descendent window
➡ Add a script to the sheet window and save it

Now you will code a Retrieve function in the Open event.

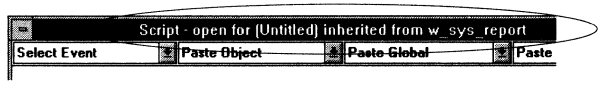
When you open this window, PowerBuilder invokes the Open event, which performs the Retrieve function. When the Retrieve function completes, PowerBuilder displays the window. To speed window display, you could code a PostEvent("post_open") function in the Open event and place the Retrieve function in the post_open event.

Retrieval arguments
If you develop applications whose DataWindow objects use retrieval arguments, you can open the sheet window using the OpenSheetWithParm function, access the arguments in the Message object, and specify the arguments in the Retrieve function.

1 Click the *Script* button in the PainterBar.

The PowerScript painter displays.

2 Make sure the title reads: *Script - Open for (Untitled) Inherited from w_sys_report.*



If the title is incorrect
If the DataWindow control was still selected, (the title ends in *::dw_sheet*), select Edit➤Select Object from the menu bar and select (Untitled).

If the event is not the Open event, pull down the Select Event listbox and select Open.

3 Type the following script:

```
dw_sheet.Retrieve()
```

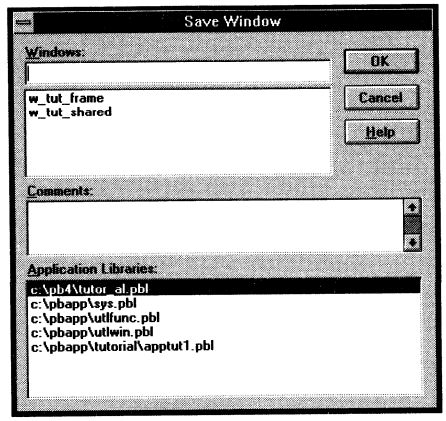
This will retrieve rows, as defined in the DataWindow object.

4 **Click the *Return* button.**
or
Select File➤Return from the menu bar.

PowerBuilder compiles your script and returns to the Window painter workspace.

5 **Select File➤Save As from the menu bar.**

The Save Window dialog box displays.



6 **Type *w_tut_report* in the Name box.**
Enter a comment in the Comment box.
Click *OK*.

PowerBuilder saves your new window as a descendant of w_sys_report.

LESSON 5

Building a Menu for the Frame Window

In MDI applications developed using the Application Library, you create a frame menu by inheriting from the m_sys_frame menu. You use descendants of m_sys_frame with windows that are descendants of application framework windows. That is, many m_sys_frame menu item scripts trigger user events defined in application framework windows.

You first create a descendent frame menu and add, modify, enable, and disable menu items, as appropriate for your application. You then use the descendent frame menu as the ancestor for all sheet menus.

In this lesson you will create a menu for the frame window by inheriting from the m_sys_frame menu. You will also add new menu items and add scripts to enable online help access. By defining application-wide items in the frame menu you avoid coding them separately in each sheet menu.

How long will this lesson take?
About 30 minutes.

What will you learn about the Application Library?

- How to create a descendant of the m_sys_frame menu. Inheriting from m_sys_frame provides a menu structure with File, Application Topics, Actions, Window, and Help menus, as well as associated toolbar buttons.
- How to define additional menu items.
- How to call Windows Help files from predefined Help menu items.

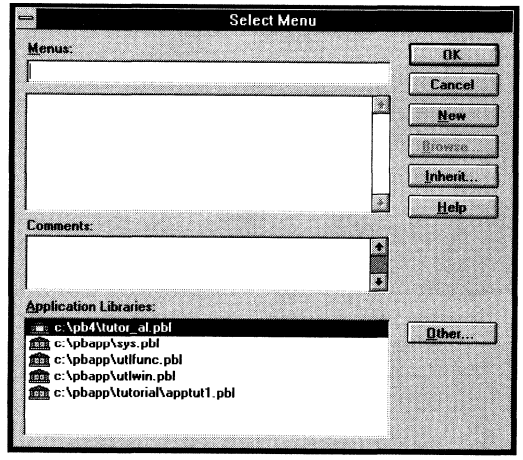
Create a descendent menu

Where you are
Lesson 5 Building a Menu for the Frame Window
➡ Create a descendent menu
Add menu items
Add more scripts
Save the menu
Add the menu to the frame window

Now you will inherit from m_sys_frame to create a frame menu.

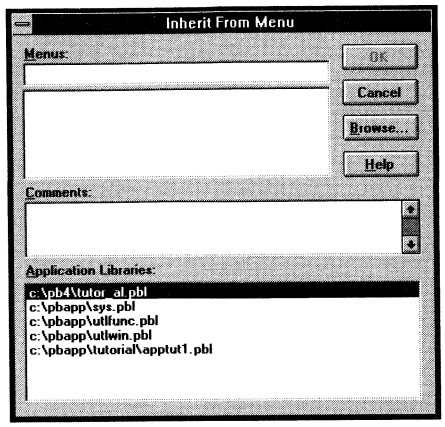
1 Click the *Menu painter* button in the PowerBar.

The Select Menu dialog box displays.



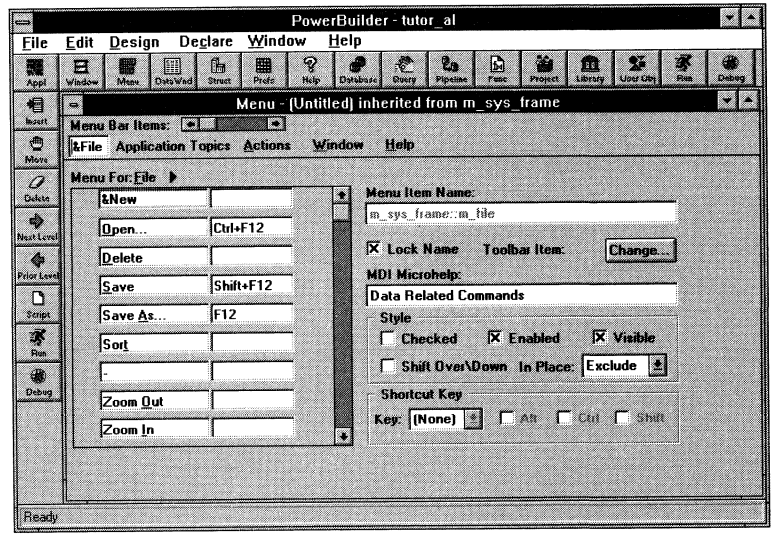
2 Click *Inherit*.

The Inherit From Menu dialog box displays.



3 Click the line that ends with *sys.pbl* in the Application Libraries box.
Select *m_sys_frame*.
Click *OK*.

The Menu painter workspace displays. The title should read: Menu - (Untitled) inherited from m_sys_frame.



Add menu items

Where you are
Lesson 5 Building a Menu for the Frame Window
Create a descendent menu
➡ Add menu items
Add more scripts
Save the menu
Add the menu to the frame window

Now you will add menu items. The m_sys_frame menu provides an Application Topics menu under which you place application-specific processes and functions. In the Application Library tutorial you add two items to the descendent menu.

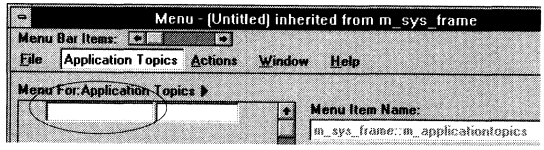
Using descendent menus
In a descendent menu, you can change menu item text but not the internal menu item name. You can add menu items to the end of a menu by typing the menu item text.

For more on menus and inheritance, see the PowerBuilder *User's Guide*.

1 Click the *Application Topics* menu bar item.

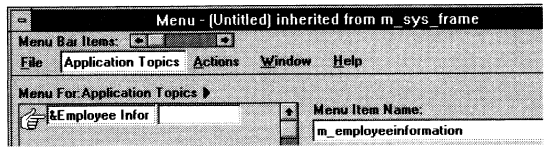
An empty box displays for the first menu item below Application Topics.

2 Click in the empty box below the Application Topics header.



3 **Type** *&Employee Information*

PowerBuilder uses this text to create the menu item name m_employeeinformation. This is the name you use in a script to refer to this menu item (for example, m_tut_frame.m_employeeinformation).



4 **Click the *Script* button in the PainterBar.**

The PowerScript painter displays.

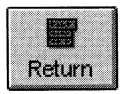
5 **Make sure the title reads:** *Script - clicked for m_employeeinformation.*

> **If the title is incorrect**
> If the event is not the Clicked event, pull down the Select Event listbox and select Clicked.

6 **Type the following script:**

```
OpenSheet(w_tut_shared,w_tut_frame,0,Cascaded!)
```

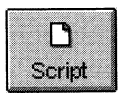
This script opens the w_tut_shared sheet window.



7 **Click the *Return* button in the PainterBar.**
or
Select File➤Return from the menu bar.

PowerBuilder compiles your script and returns to the Menu painter workspace.

8 **Click in the empty box below Employee Information.**
Type *&Total Compensation Report*



9 **Click the *Script* button in the PainterBar.**

The PowerScript painter displays.

10 **Make sure the title reads:** *Script - clicked for m_totalcompensationreport.*

> **If the title is incorrect**
> If the event is not the Clicked event, pull down the Select Event listbox and select Clicked.

11 **Type the following script:**

```
OpenSheet(w_tut_report,w_tut_frame,0,Cascaded!)
```

This script opens the w_tut_report sheet window.

12 **Click the *Return* button in the PainterBar.**
or
Select File➤Return from the menu bar.

PowerBuilder compiles your script and returns to the Menu painter workspace.

Add more scripts

Where you are
Lesson 5 Building a Menu for the Frame Window
Create a descendent menu
Add menu items
➡ Add more scripts
Save the menu
Add the menu to the frame window

Now you will add scripts for m_sys_frame Help menu items. You use these items in conjunction with Windows online Help files.

1 **Click the *Help* menu bar item.**
Click the *Contents* menu item.
Click the *Script* button in the PainterBar.

The PowerScript painter displays.

2 **Make sure the title reads: *Script - clicked for m_sys_frame::m_helpindex.***

If the title is incorrect
If the event is not the Clicked event, pull down the Select Event listbox and select Clicked.

3 **Type the following script:**

```
ShowHelp("c:\pbapp\tutorial\tutor_al.hlp", &
   Index!)
```

This starts the Windows Help system and displays the contents window for the online Help provided with the Application Library tutorial application. This file has usage information as well as information on the objects used in the application.

Tutor_al.hlp must be accessible
If you installed the Application Library tutorial files in a directory other than \PBAPP\TUTORIAL, be sure to specify the appropriate path. Alternatively, you could ensure that the tutor_al.hlp file is in a directory named in the DOS PATH statement.

4 **Click the *Return* button in the PainterBar.**
or
Select File➤Return from the menu bar..

PowerBuilder compiles your script and returns to the Menu painter workspace.

5 **Click the *Search for Help On* menu item.**
Click the *Script* button in the PainterBar.

The PowerScript painter displays.

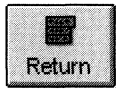
6 **Make sure the title reads:** *Script - clicked for m_sys_frame::m_search.*

> **If the title is incorrect**
> If the event is not the Clicked event, pull down the Select Event listbox and select Clicked.

7 **Type the following script:**

```
ShowHelp("c:\pbapp\tutorial\tutor_al.hlp", &
   Keyword!,"")
```

This starts the Windows Help system using the online Help provided with the Application Library tutorial application and displays the Search dialog box.



8 **Click the *Return* button in the PainterBar.**
or
Select File➤Return from the menu bar.

PowerBuilder compiles your script and returns to the Menu painter workspace.

9 **Click the *How to use Help* menu item.**
Click the *Script* button in the PainterBar.

The PowerScript painter displays.

10 Make sure the title reads: *Script - clicked for m_sys_frame::m_helponhelp.*

> **If the title is incorrect**
> If the event is not the Clicked event, pull down the Select Event listbox and select Clicked.

11 Type the following script:

```
ShowHelp("winhelp.hlp",Index!)
```

This starts the Windows Help system displaying the file that explains how to use the Help system.

12 Click the *Return* **button in the PainterBar.**
or
Select File➤Return from the menu bar.

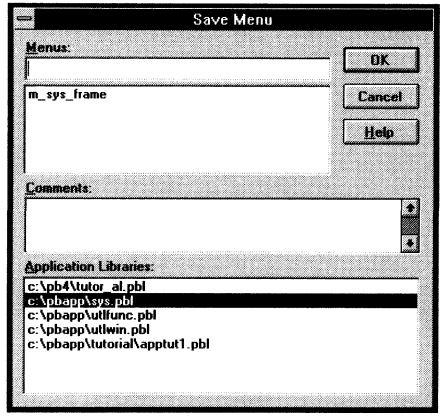
PowerBuilder compiles your script and returns to the Menu painter workspace.

Save the menu

Where you are
Lesson 5 Building a Menu for the Frame Window
Create a descendent menu
Add menu items
Add more scripts
➡ Save the menu
Add the menu to the frame window

1 Select File➤Save As from the menu bar.

The Save Menu dialog box displays.



2 Click the line that ends with *tutor_al.pbl* in the Application Libraries box.

This makes sure that the menu is stored in your application's library.

3 Type *m_tut_frame* in the Menus box.
Add a comment in the Comments box.
Click *OK*.

PowerBuilder saves your new menu as a descendant of m_sys_frame.

4 Select File➤Close from the menu bar.

PowerBuilder closes the Menu painter.

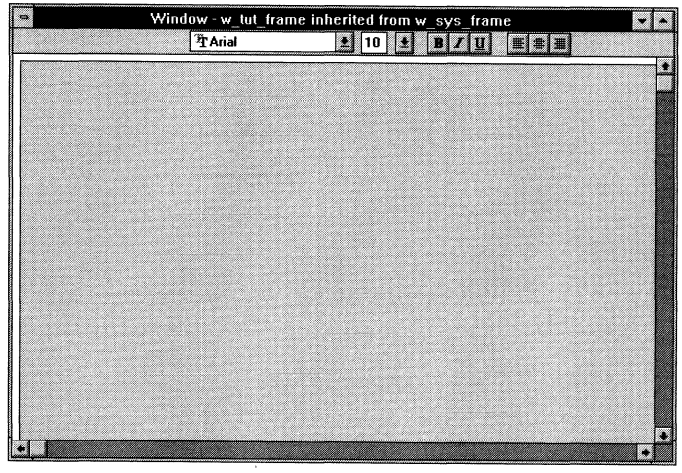
Add the menu to the frame window

Where you are
Lesson 5 Building a Menu for the Frame Window
Create a descendent menu
Add menu items
Add more scripts
Save the menu
➡ Add the menu to the frame window

Now you will use the Window painter to associate a menu with a window.

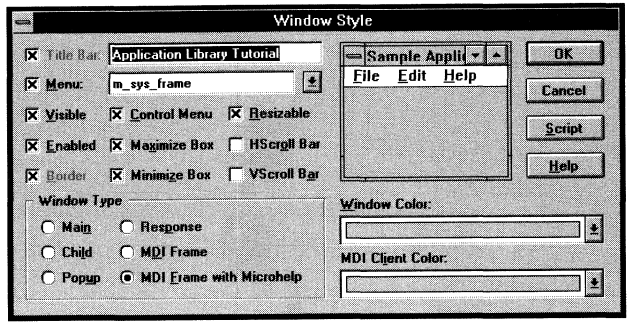
1 Make sure you are in the Window painter with the w_tut_frame window displayed.
If you are not, open the Window painter and select the *w_tut_frame* window.

Use the list of most recently accessed objects
To display one of the four most recently accessed objects, click on the File menu and select the object from the list displayed at the bottom of the menu.

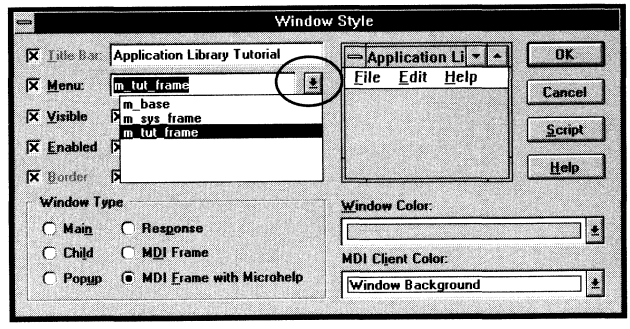


2 Select Design➤Window Style from the menu bar.

The Window Style dialog box displays.



3 Click the down arrow in the dropdown listbox.
Click *m_tut_frame*.
Click *OK*.



The Window painter workspace displays.

4 Select File➤Close from the menu bar.

5 Click *Yes* **to save changes.**

LESSON 6

Building a Menu for the First Sheet Window

In MDI applications, sheet menus typically inherit from the related frame menu and add, modify, enable, and disable items as needed. This provides ease of maintenance, because application-wide changes made to the frame menu are inherited automatically by the sheet menus.

In this lesson you will create a menu for the first sheet window by inheriting from the m_tut_frame menu. You will also enable menu items that apply to the first sheet window (such as print, close, insert row, and delete row) and disable the menu item that opens the first sheet window (since only one instance can be open at a time).

How long will this lesson take?
About 30 minutes.

What will you learn about the Application Library?

- How to create a descendant of the frame menu. Inheriting from the frame menu (which in turn is a descendant of m_sys_frame) provides m_sys_frame functionality as well as the application-specific customization you added to the m_tut_frame menu. For example, you automatically have access to toolbar buttons (defined in m_sys_frame) and online Help (defined in m_tut_frame).
- How to modify menu items. Many of the predefined menu items are hidden but you can make them visible as needed.
- How to enable descendent menu items and toolbar buttons.

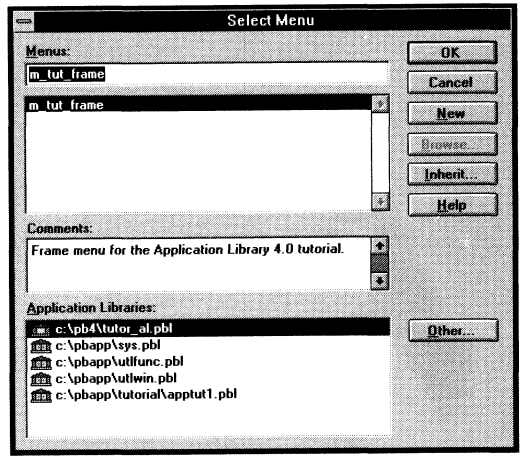
Create a descendent menu

Where you are
Lesson 6 Building a Menu for the First Sheet Window
➡ Create a descendent menu
Modify menu items
Save the menu
Add the menu to the first sheet window
Run the application

Now you will create a sheet menu by inheriting from m_tut_frame, the frame menu used by the Application Library tutorial.

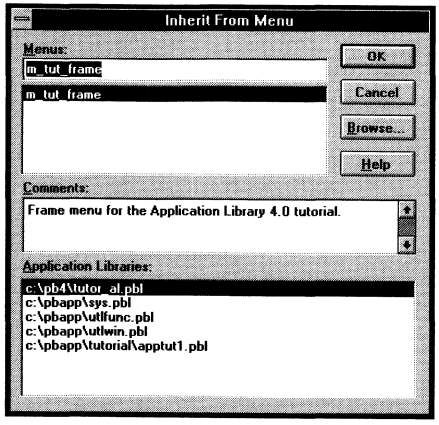
1 Click the *Menu painter* button in the PowerBar.

The Select Menu dialog box displays.



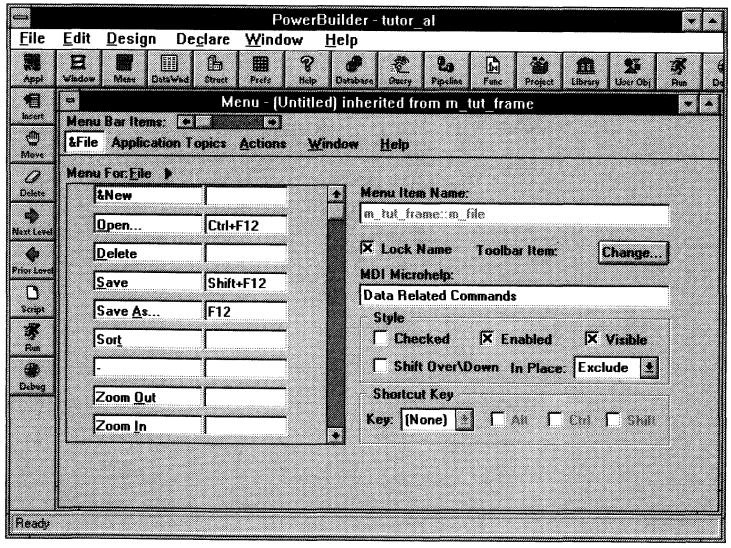
2 Click *Inherit.*

The Inherit From Menu dialog box displays.



3 Select *m_tut_frame.*
Click *OK.*

The Menu painter workspace displays. The title should read Menu - (Untitled) inherited from m_tut_frame.



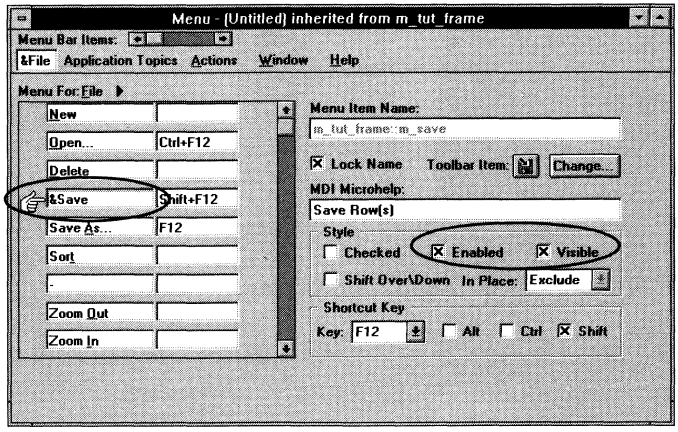
Modify menu items

Where you are
Lesson 6 Building a Menu for the First Sheet Window
Create a descendent menu
➡ Modify menu items
Save the menu
Add the menu to the first sheet window
Run the application

Now you will enable menu items and toolbar buttons that are relevant to the sheet window. Additionally, because this application has an arbitrary rule that only one sheet instance can be displayed at a time, you will disable display of the Employee Information menu item.

Most items are disabled and invisible
Because m_sys_frame must provide functionality for all windows in the application framework, there are many m_sys_frame menu items whose initial state is disabled and invisible. For example, there are many items in the File menu but only Print Setup and Exit are enabled initially.

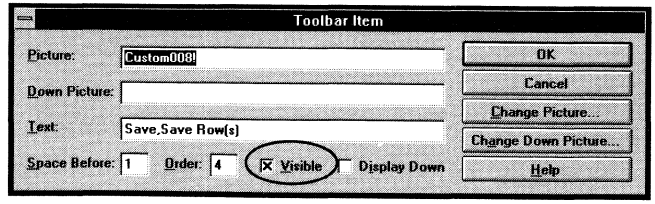
1 **Click the *File* menu bar item.**
Click the *Save* menu item.
Select the *Enabled* checkbox.
Select the *Visible* checkbox.



2 **Click** *Change.*

The Toolbar Item dialog box displays.

3 **Select the** *Visible* **checkbox to enable display of the Save button.**



4 **Click** *OK.*

The Menu painter workspace displays.

5 **Click the** *Print* **menu item (you will have to scroll down the list).**
Select the *Enabled* **checkbox.**
Select the *Visible* **checkbox.**

6 **Click** *Change.*

The Toolbar Item dialog box displays.

7 **Select the** *Visible* **checkbox to enable display of the Print button.**
Click *OK.*

The Menu painter workspace displays.

8 **Click the** *Close* **menu item.**
Select the *Enabled* **checkbox.**
Select the *Visible* **checkbox.**

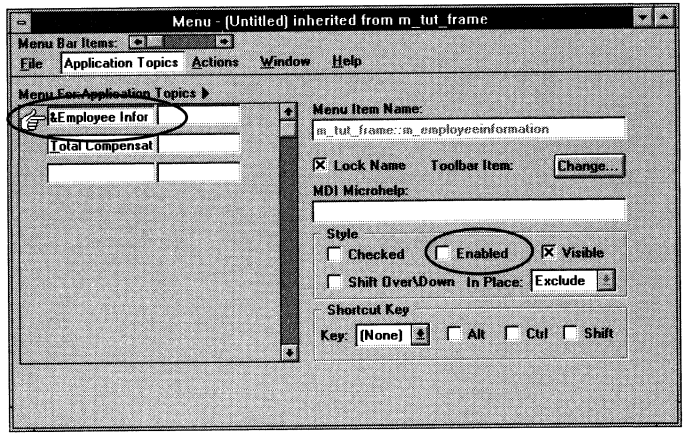
9 **Click** *Change.*

The Toolbar Item dialog box displays.

10 **Select the** *Visible* **checkbox to enable display of the Close button.**
Click *OK.*

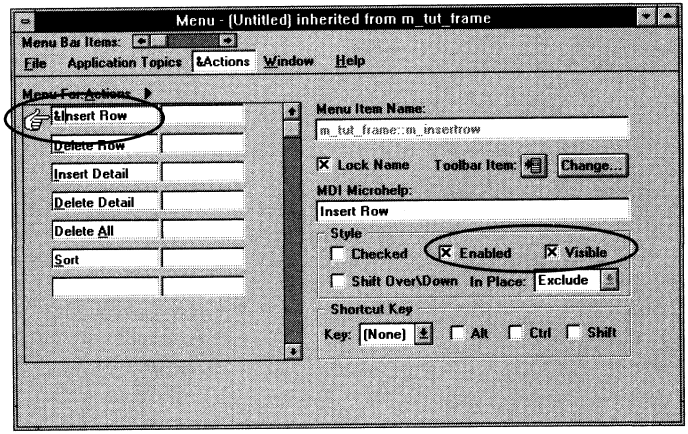
The Menu painter workspace displays.

11 **Click the** *Application Topics* **menu bar item.**
Click the *Employee Information* **menu item.**
Deselect the *Enabled* **checkbox.**



12 **Click the** *Actions* **menu bar item.**
Select the *Enabled* **checkbox.**

13 **Click the** *Insert Row* **menu item (not the Insert Detail menu item).**
Select the *Enabled* **checkbox.**
Select the *Visible* **checkbox.**



14 **Click** *Change.*

The Toolbar Item dialog box displays.

15 **Select the** *Visible* **checkbox to enable display of the Insert Row button.**
Click *OK.*

The Menu painter workspace displays.

16 **Click the** *Delete Row* **menu item.**
Select the *Enabled* **checkbox.**
Select the *Visible* **checkbox.**

17 **Click** *Change.*

The Toolbar Item dialog box displays.

18 **Select the** *Visible* **checkbox to enable display of the Delete Row button.**
Click *OK.*

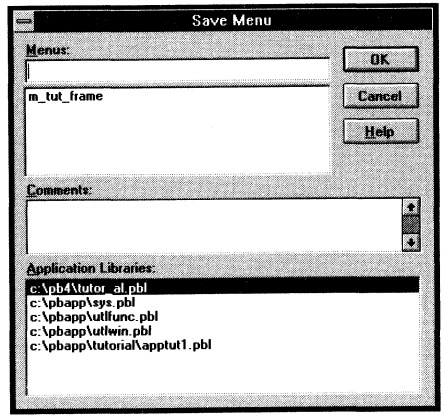
The Menu painter workspace displays.

Save the menu

Where you are
Lesson 6 Building a Menu for the First Sheet Window
Create a descendent menu
Modify menu items
➡ Save the menu
Add the menu to the first sheet window
Run the application

1 Select File➤Save As from the menu bar.

The Save Menu dialog box displays.



2 Type *m_tut_shared* in the Menus box.
Add a comment in the Comments box.
Click *OK*.

PowerBuilder saves your new menu as a descendant of m_tut_frame.

3 Select File➤Close from the menu bar.

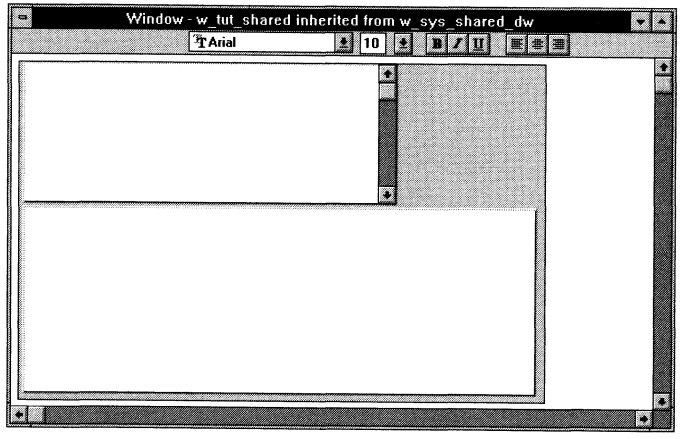
PowerBuilder closes the Menu painter.

Add the menu to the first sheet window

Where you are
Lesson 6 Building a Menu for the First Sheet Window
Create a descendent menu
Modify menu items
Save the menu
➡ Add the menu to the first sheet window
Run the application

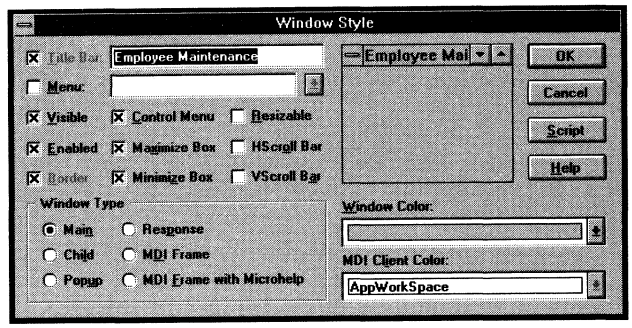
Now you will use the Window painter to associate the menu you just created with the first sheet window.

1 **Make sure you are in the Window painter with the w_tut_shared window displayed.**
If you are not, open the Window painter and select the *w_tut_shared* window.

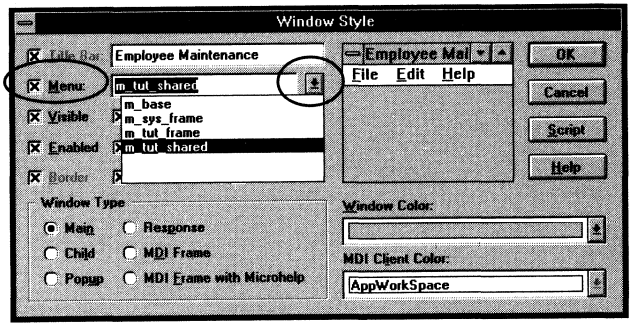


2 Select Design➤Window Style from the menu bar.

The Window Style dialog box displays.



3 Select the *Menu* box.
Click the down arrow in the dropdown listbox.
Click *m_tut_shared*.
Click *OK*.



The Window painter workspace displays.

4 Select File➤Close from the menu bar.

5 Click *Yes* to save changes.

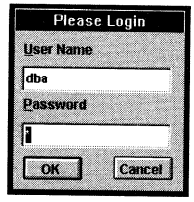
Run the application

Where you are
Lesson 6 Building a Menu for the First Sheet Window
Create a descendent menu
Modify menu items
Save the menu
Add the menu to the first sheet window
➡ Run the application

Now you will verify that the menus display properly by running the Application Library tutorial.

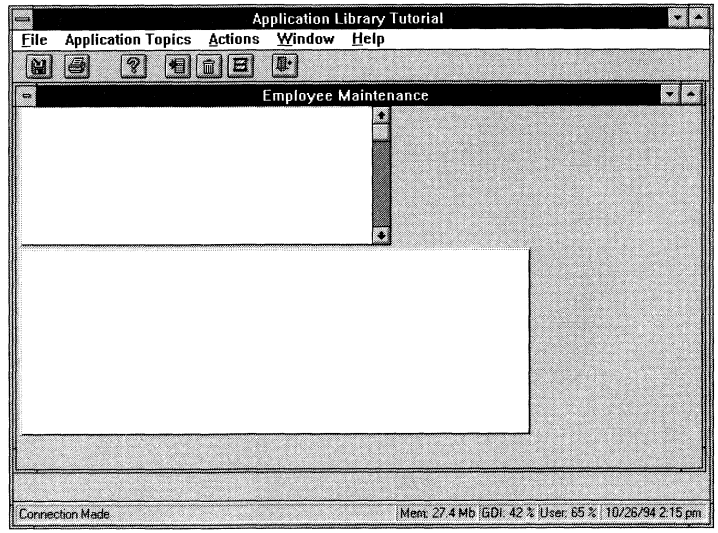
1 **Click the *Run* button in the PowerBar.**
or
Select File➢Run from the menu bar.
If PowerBuilder prompts you to save changes, click *Yes*.

The frame window's Open event calls the f_login function, which displays the w_login dialog box.



2 **Type *sql* (or whatever password you specified in the tutor_al.ini file). Click *OK*.**

The application connects to the database and displays the Employee Maintenance sheet window, including the associated menu bar and toolbar.



3 **Click on the various menus to make sure that they display correctly. The Help menu items should work; all other menu items require DataWindow objects, which you have not yet associated with the DataWindow controls.**

You will associate DataWindow objects with this window's DataWindow controls in Lesson 8.

4 **Exit the application by pressing** ALT+F4.

LESSON 7

Building a Menu for the Second Sheet Window

In this lesson you will create a menu for the second sheet window by inheriting from the m_tut_frame menu. You will also enable menu items that apply to the second sheet window (such as print zoom and print preview) and disable the menu item that opens the second sheet window (since only one instance can be open at a time).

How long will this lesson take?
About 10 minutes.

What will you learn about the Application Library?

- How to create a descendant of the frame menu. Inheriting from the frame menu (which in turn is a descendant of m_sys_frame) provides m_sys_frame functionality as well as the application-specific customization you added to the m_tut_frame menu. For example, you automatically have access to toolbar buttons (enabled in m_sys_frame) and online Help (defined in m_tut_frame).
- How to enable descendent menu items and toolbar buttons.

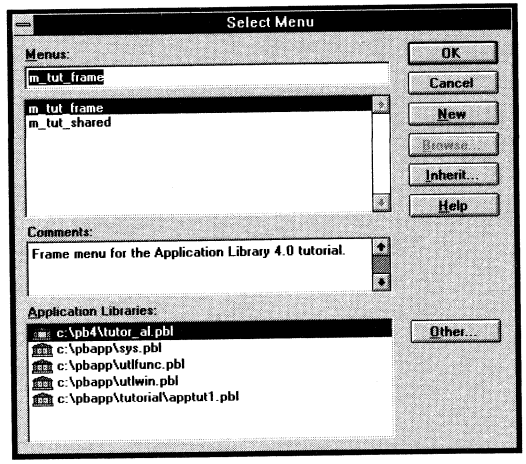
Create a descendent menu

Where you are
Lesson 7 Building a Menu for the Second Sheet Window
➡ Create a descendent menu
Modify menu items
Save the menu
Add the menu to the second sheet window
Run the application

Now you will create another sheet menu by inheriting from m_tut_frame, the frame menu used by the Application Library tutorial.

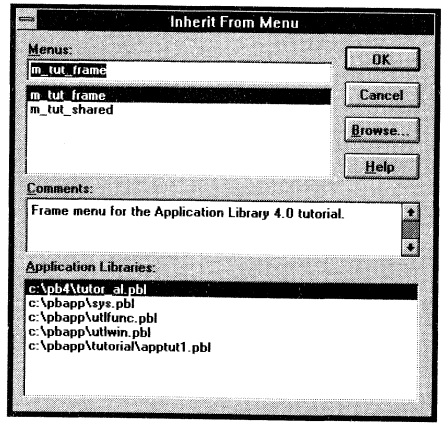
1 Click the *Menu painter* button in the PowerBar.

The Select Menu dialog box displays.



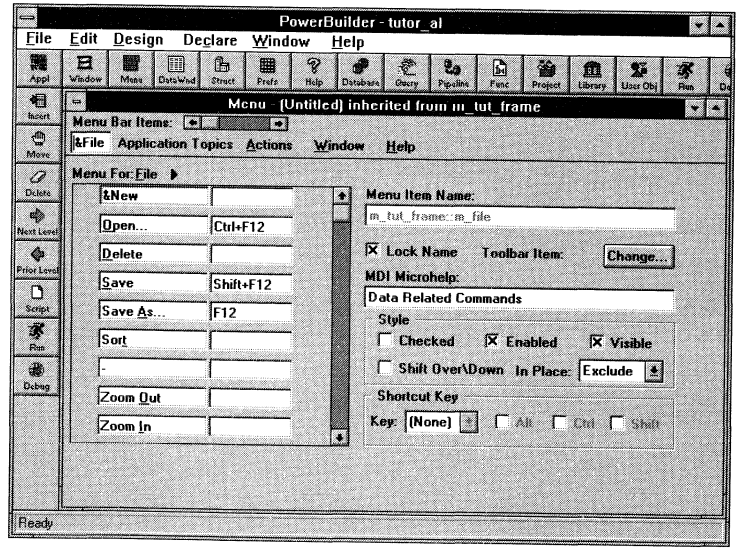
2 Click *Inherit.*

The Inherit From Menu dialog box displays.



3 Select *m_tut_frame.*
Click *OK.*

The Menu painter workspace displays. The title should read: Menu - (Untitled) inherited from m_tut_frame.

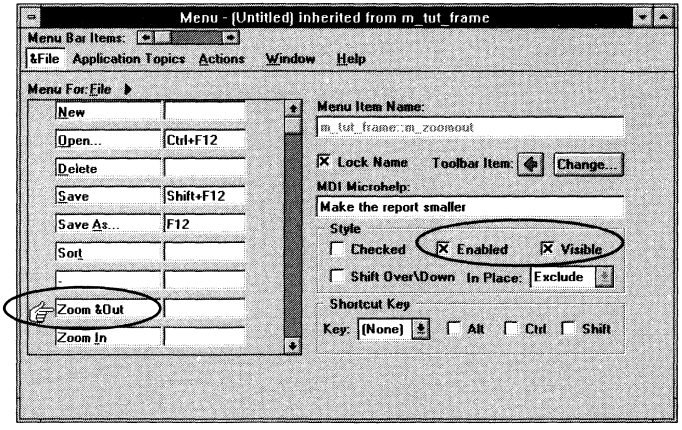


Modify menu items

Where you are
Lesson 7 Building a Menu for the Second Sheet Window
Create a descendent menu
➡ Modify menu items
Save the menu
Add the menu to the second sheet window
Run the application

Now you will enable menu items and toolbar buttons that are relevant to this sheet window. Additionally, because this application has an arbitrary rule that only one sheet instance can be displayed at a time, you will disable display of the Total Compensation Report menu item.

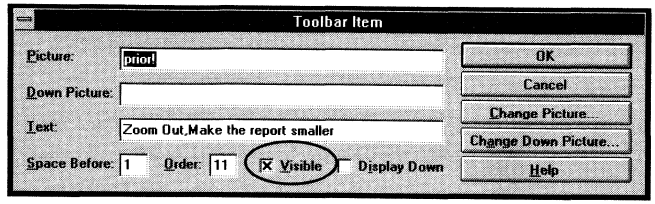
1 **Click the *File* menu bar item.**
Click the *Zoom Out* menu item.
Select the *Enabled* checkbox.
Select the *Visible* checkbox.



2 **Click *Change*.**

The Toolbar Item dialog box displays.

3 **Select the *Visible* checkbox to enable display of the Zoom Out button.**



4 **Click *OK*.**

The Menu painter workspace displays.

5 **Click the *Zoom In* menu item.**
Select the *Enabled* checkbox.
Select the *Visible* checkbox.

6 **Click *Change*.**

The Toolbar Item dialog box displays.

7 **Select the *Visible* checkbox to enable display of the Zoom In button.**
Click *OK*.

The Menu painter workspace displays.

8 **Click the *Print* menu item (you will have to scroll down the list).**
Select the *Enabled* checkbox.
Select the *Visible* checkbox.

9 **Click *Change* to enable display of the Print button.**

The Toolbar Item dialog box displays.

10 **Select the *Visible* checkbox.**
Click *OK*.

The Menu painter workspace displays.

11 Click the *Print Preview* menu item.
Select the *Enabled* checkbox.
Select the *Visible* checkbox.

12 Click the *Close* menu item.
Select the *Enabled* checkbox.
Select the *Visible* checkbox.

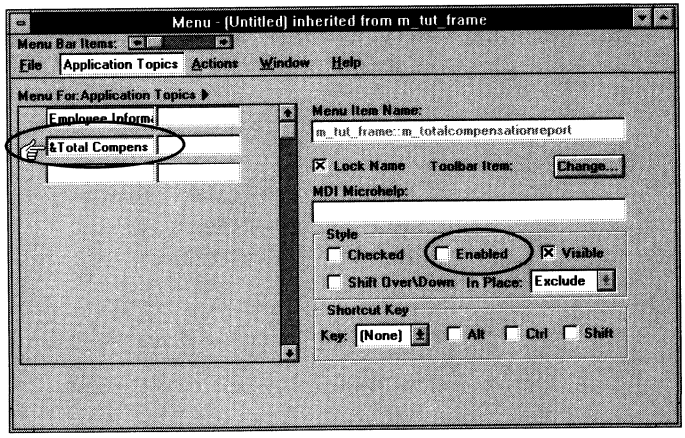
13 Click *Change*.

The Toolbar Item dialog box displays.

14 Select the *Visible* checkbox to enable display of the Close button.
Click *OK*.

The Menu painter workspace displays.

15 Click the *Application Topics* menu bar item.
Click the *Total Compensation Report* menu item.
Deselect the *Enabled* checkbox.

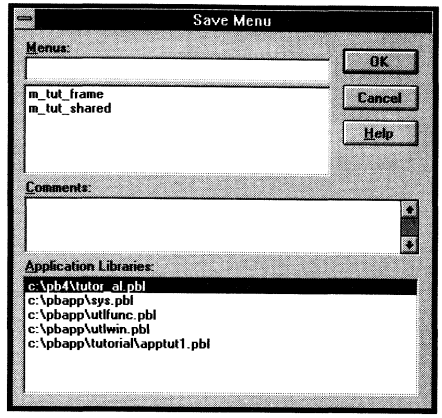


Save the menu

Where you are
Lesson 7 Building a Menu for the Second Sheet Window
Create a descendent menu
Modify menu items
➡ Save the menu
Add the menu to the second sheet window
Run the application

1 Select File➤Save As from the menu bar.

The Save Menu dialog box displays.



2 Type *m_tut_report* in the Menus box.
Add a comment in the Comments box.
Click *OK*.

PowerBuilder saves your new menu as a descendant of m_tut_frame.

3 Select File➤Close from the menu bar.

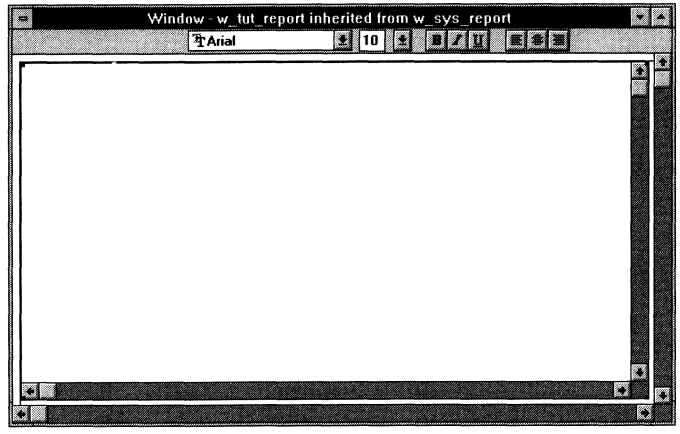
PowerBuilder closes the Menu painter.

Add the menu to the second sheet window

Where you are
Lesson 7 Building a Menu for the Second Sheet Window
Create a descendent menu
Modify menu items
Save the menu
➡ Add the menu to the second sheet window
Run the application

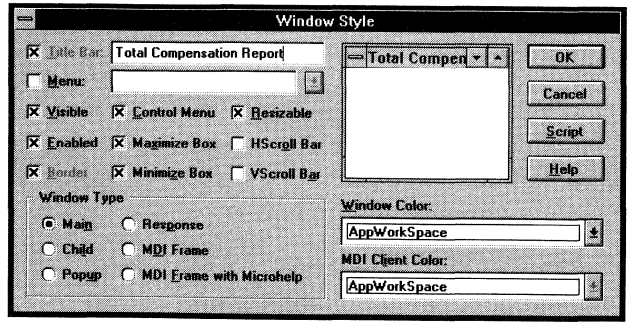
Now you will use the Window painter to associate the menu you just created with the second sheet window.

1 **Make sure you are in the Window painter with the w_tut_report window displayed.**
If you are not, open the Window painter and select the *w_tut_report* window.

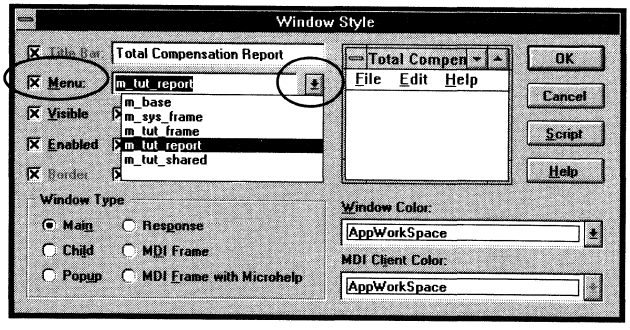


2 Select Design➤Window Style from the menu bar.

The Window Style dialog box displays.



3 Select the *Menu* box.
Click the down arrow in the dropdown listbox.
Click *m_tut_report*.
Click *OK*.



The Window painter workspace displays.

4 Select File➤Close from the menu bar.

5 Click *Yes* to save changes.

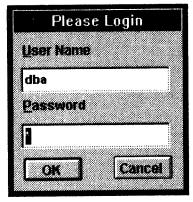
Run the application

Where you are
Lesson 7 Building a Menu for the Second Sheet Window
Create a descendent menu
Modify menu items
Save the menu
Add the menu to the second sheet window
➡ Run the application

Now you will verify that the menu displays properly by running the Application Library tutorial.

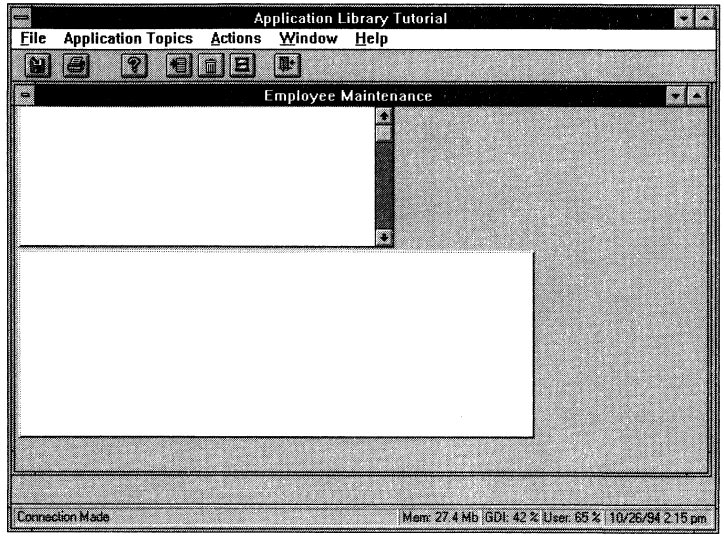
1 **Click the *Run* button in the PowerBar.**
or
Select File➤Run from the menu bar.
If PowerBuilder prompts you to save changes, click *Yes*.

The frame window's Open event calls the f_login function, which displays the w_login dialog box.



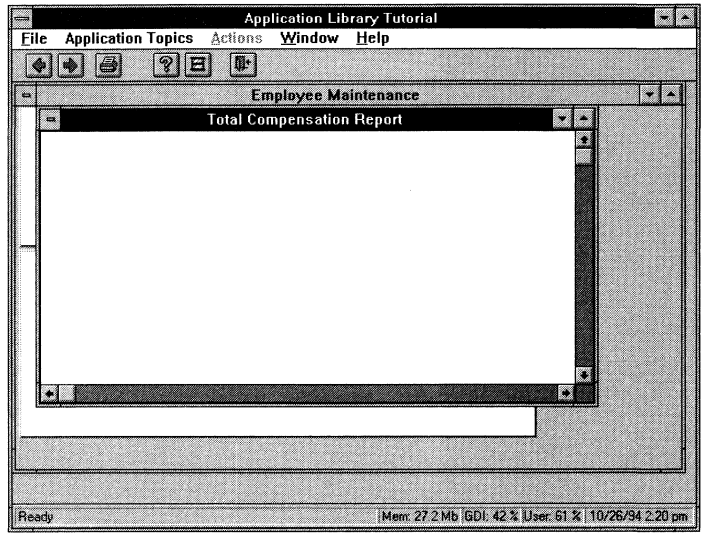
2 Type *sql* (or whatever password you specified in the tutor_al.ini file). Click *OK*.

The application connects to the database and displays the first sheet window.



3 **Select Application Topics➤Total Compensation Report from the menu bar.**

The Total Compensation Report sheet should display.



4 **Click on the various menus to make sure that they display correctly. The Help menu items should work; all other menu items require a DataWindow object, which you have not yet associated with the window's DataWindow control.**

You will associate a DataWindow object with the sheet window's DataWindow control in Lesson 8.

5 **Exit the application by pressing** ALT+F4.

LESSON 8

Associating DataWindow Objects with DataWindow Controls

Application Library application framework windows include DataWindow controls with encapsulated scripts that perform basic window management and database functions. Before you can use the window to access data, however, you must associate each DataWindow *control* with a DataWindow *object*. After associating DataWindow objects with DataWindow controls, you can create additional scripts to meet your application's needs.

In this lesson you will associate predefined DataWindow objects with the DataWindow controls in the w_tut_shared and w_tut_report windows. You will also use Application library functions to perform basic activity logging in the w_tut_shared window's on_insert, on_update, and on_delete user events.

How long will this lesson take?
About 35 minutes.

What will you learn about the Application Library?

- **The dw_sheet DataWindow control in w_tut_shared** is inherited from the uo_dw user object.
- **The uo_dw user object** provides many encapsulated events and functions, including RowFocusChanged, DBError, ItemFocusChanged, and SQLPreview.
- **The encapsulated DBError event script** calls the f_error_box function, which in many cases eliminates the need for database error checking.
- **The encapsulated SQLPreview event script** calls the on_insert, on_update, and on_delete user events, which allows you to perform optional processing just prior to accessing the database.
- **The f_write_log function** appends a string to the end of a specified file.

Select DataWindow objects for w_tut_shared

Where you are
Lesson 8 Associating DataWindow Objects with DataWindow Controls
➡ Select DataWindow objects for w_tut_shared
Add scripts for the dw_sheet DataWindow control
Select DataWindow object for w_tut_report
Run the application

Now you will use the Window painter to associate the d_emplist and d_employee DataWindow objects with the dw_sheet and dw_detail DataWindow controls in the w_tut_shared window.

Multiple levels of inheritance
It may help to know the levels of inheritance in this situation:

uo_dw standard DataWindow user object

↳w_sys_single_dw window

↳w_sys_multi_dw window

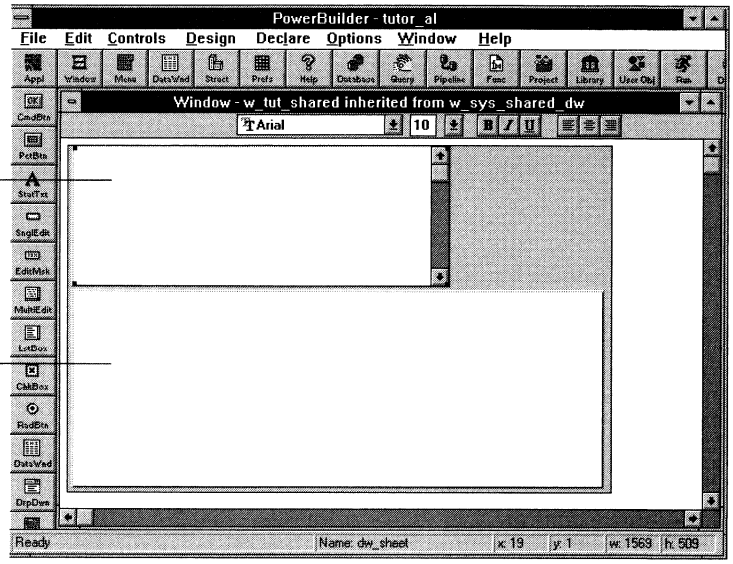
↳w_sys_shared_dw window

↳w_tut_shared window

Each step in the inheritance chain further customizes the DataWindow control's behavior. For example:

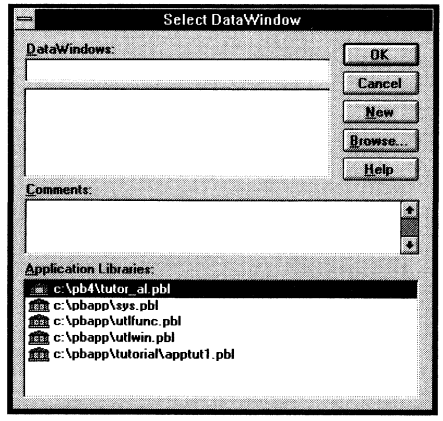
- Uo_dw includes support for the DBError event.
- W_sys_single_dw adds support for the ItemFocusChanged event.
- W_sys_multi_dw adds support for the RowFocusChanged event.
- W_sys_shared_dw adds support for shared DataWindows.
- And you will add support in w_tut_shared for the on_insert, on_update, and on_delete user events.

1 **Make sure you are in the Window painter with the w_tut_shared window displayed.**
If you are not, open the Window painter and select the *w_tut_shared* window.



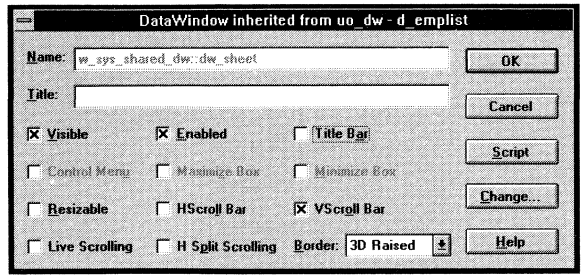
2 **Double-click inside the top DataWindow control (dw_sheet).**

The Select DataWindow dialog box displays.



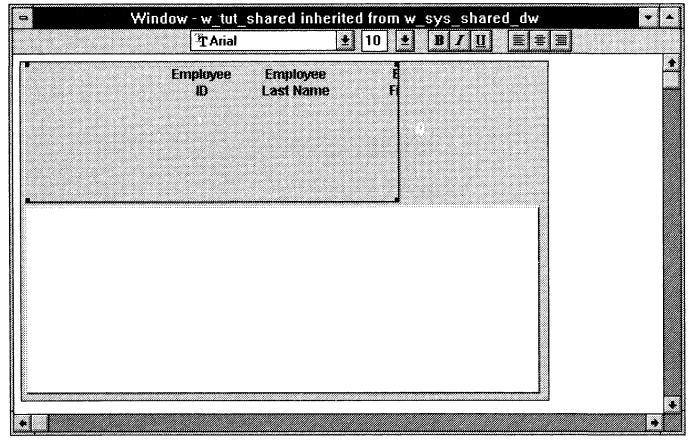
3 Click the line that ends with *apptut1.pbl* in the Application Libraries box. Double-click *d_emplist*.

The DataWindow dialog box displays some of the attributes of the dw_sheet DataWindow control.



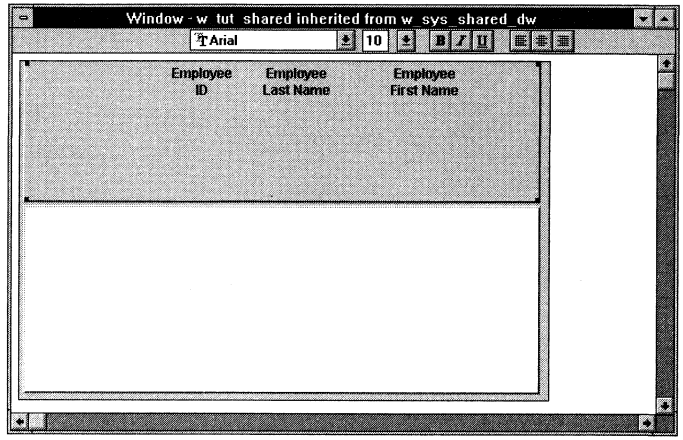
4 Click *OK*.

The Window painter workspace displays.



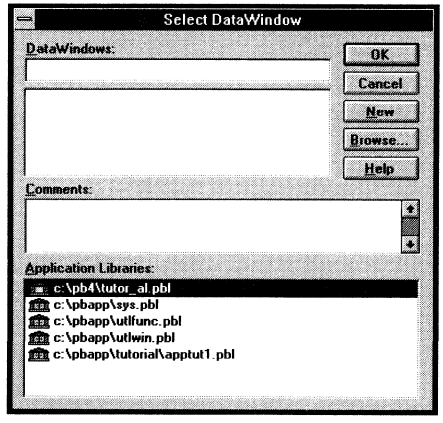
5 Adjust the size of the DataWindow control if necessary.

If you do not see all the columns in the DataWindow control, make it wider.



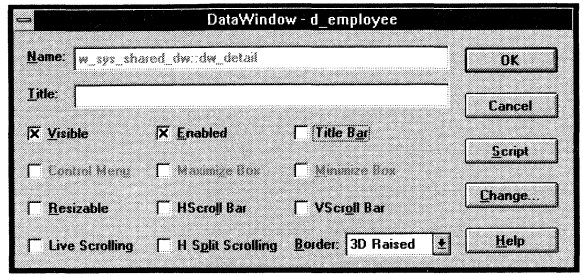
6 Double-click inside the bottom DataWindow control (dw_detail).

The Select DataWindow dialog box appears.



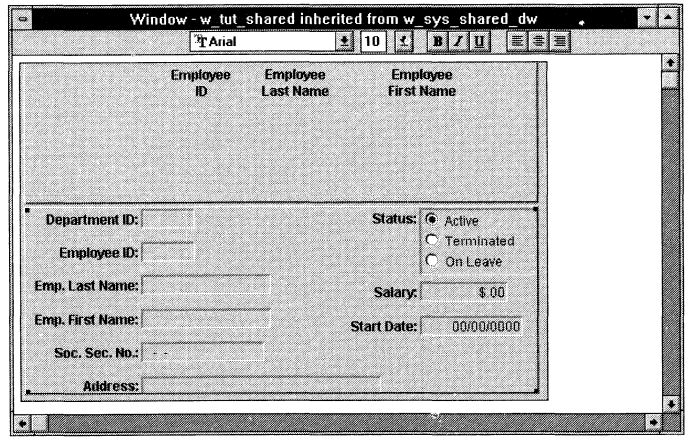
7 Click the line that ends with *apptut1.pbl* in the Application Libraries box. Double-click *d_employee*.

The DataWindow dialog box displays some of the attributes of the dw_detail DataWindow control.



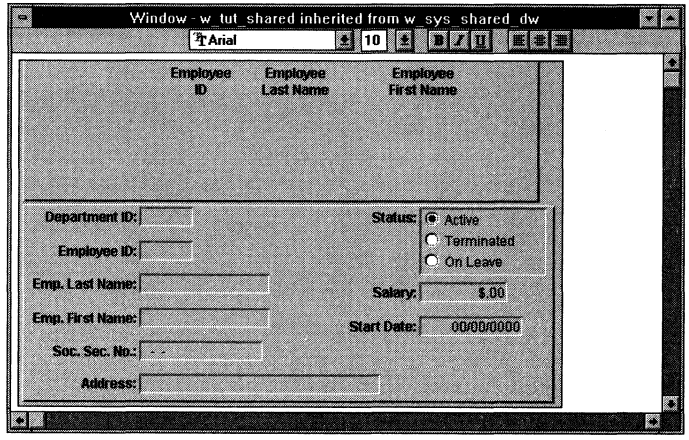
8 Click OK.

The Window painter displays.



9 Adjust the size of the DataWindow control if necessary.

If you do not see all the columns in the DataWindow control, make it bigger (you may need to make the window bigger too).



Add scripts for the dw_sheet DataWindow control

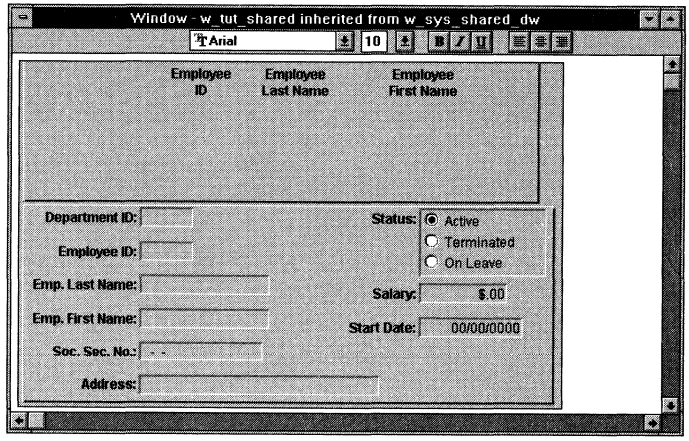
Where you are
Lesson 8 Associating DataWindow Objects with DataWindow Controls
Select DataWindow objects for w_tut_shared
➡ Add scripts for the dw_sheet DataWindow control
Select DataWindow object for w_tut_report
Run the application

Now you will create a window function to log database updates and call it from three different user events. The SQLPreview event script encapsulated in the dw_sheet DataWindow control triggers different user events, depending on the update type:

Update type	User event triggered by SQLPreview
Update existing row	on_update
Update new row	on_insert
Delete row	on_delete

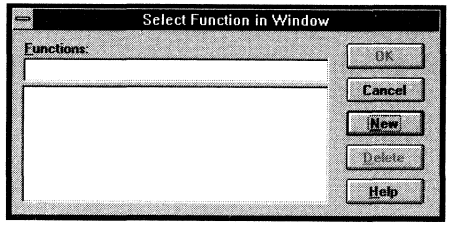
The Application Library tutorial application uses these user events to invoke a window function that writes database request information to a log file. First you will code the window function and then you will code the user events.

1 **Make sure you are in the Window painter with the w_tut_shared window displayed.**
If you are not, open the Window painter and select the *w_tut_shared* window.



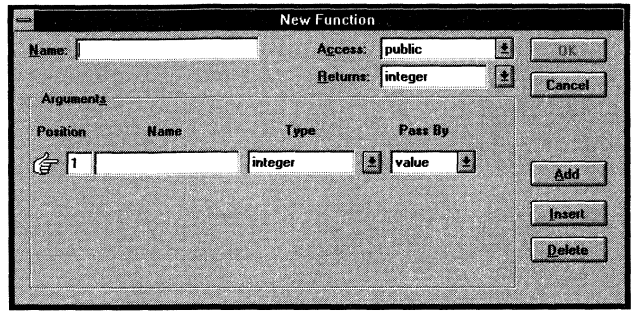
2 **Select Declare➤Window Functions from the menu bar.**

The Select Function in Window dialog box displays.



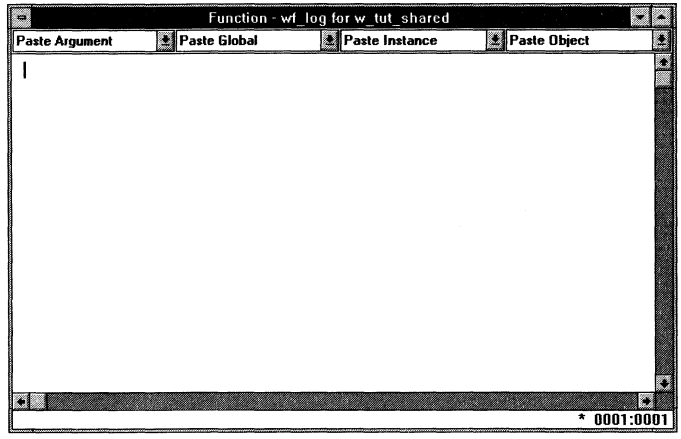
3 Click *New*.

The New Function dialog box displays.



4 Type *wf_log* in the Name box.
Type *Boolean* in the Returns box.
Type *database_action* in the Name box.
Type *string* in the Type box.
Click *OK*.

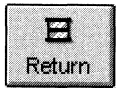
The PowerScript painter displays.



5 **Select File➤Import from the menu bar and select wf_log.txt from the \PBAPP\TUTORIAL directory.**
or
Type the following script:

```
// Write to log file.
// Uses f_write_log function.

dwbuffer dwb_buf
long     ll_row
string   ls_log_text
string   ls_divider = "======================="
string   ls_log_file = "c:\pb4\tutor_al.log"
integer  li_emp_id

dw_sheet.GetUpdateStatus(ll_row,dwb_buf)
li_emp_id = dw_sheet.GetItemNumber(ll_row, &
   "emp_id",dwb_buf,FALSE)
ls_log_text = "Employee ID: "  &
   + String(li_emp_id) + " "  &
   + database_action + ", "  &
   + String(Today()) + ", "  &
   + String(Now())

IF NOT f_write_log(ls_log_file,ls_divider) THEN
   Return FALSE
END IF
IF NOT f_write_log(ls_log_file,ls_log_text) THEN
   Return FALSE
END IF
Return TRUE
```

You may need to change the path for the ls_log_file variable.



6 **Click the *Return* button.**
or
Select File➤Return from the menu bar.

PowerBuilder compiles your function and returns to the Window painter workspace.

7 **Move the pointer to an unused area in the top DataWindow control and right-click.**

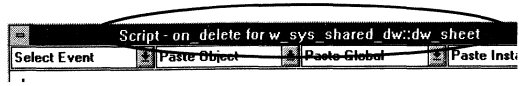
This selects the DataWindow control and displays the popup menu.

8 **Select *Script* from the popup menu.**

The PowerScript painter displays.

9 Click the down arrow next to the Select Event listbox and select *on_delete*.

The title should read: Script - on_delete for w_sys_shared::dw_sheet.



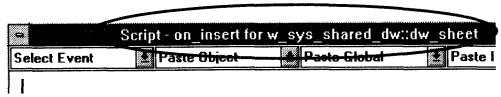
If the title is incorrect
If the object is not dw_sheet, select Edit➤Select Object from the menu bar and select dw_sheet.

10 Type the following script:

```
IF NOT wf_log("Delete") THEN
   f_error_box("Write Log", &
      "Problem writing to log file.")
   this.SetActionCode(1)
END IF
```

11 Click the down arrow next to the Select Event listbox and select *on_insert*.

The title should read: Script - on_insert for w_sys_shared::dw_sheet.

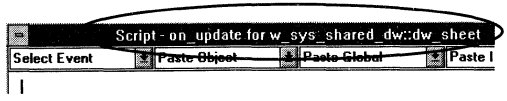


12 Type the following script:

```
IF NOT wf_log("Insert") THEN
   f_error_box("Write Log", &
      "Problem writing to log file.")
   this.SetActionCode(1)
END IF
```

13 Click on the down arrow next to the Select Event listbox and select *on_update*.

The title should read: Script - on_update for w_sys_shared::dw_sheet.



14 Type the following script:

```
IF NOT wf_log("Update") THEN
   f_error_box("Write Log", &
      "Problem writing to log file.")
   this.SetActionCode(1)
END IF
```

15 Click the *Return* button.
or
Select File➤Return from the menu bar.

PowerBuilder compiles your scripts and returns to the Window painter workspace.

16 Select File➤Save from the menu bar.

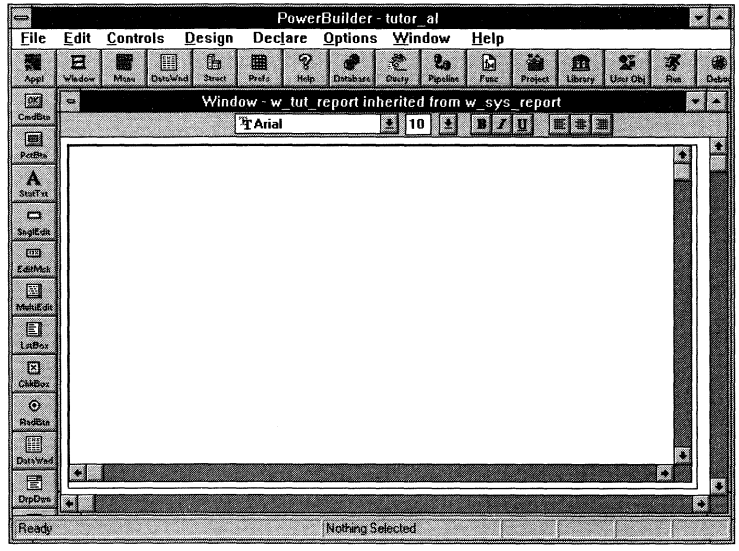
PowerBuilder saves your changes.

Select DataWindow object for w_tut_report

Where you are
Lesson 8 Associating DataWindow Objects with DataWindow Controls
Select DataWindow objects for w_tut_shared
Add scripts for the dw_sheet DataWindow control
➡ Select DataWindow object for w_tut_report
Run the application

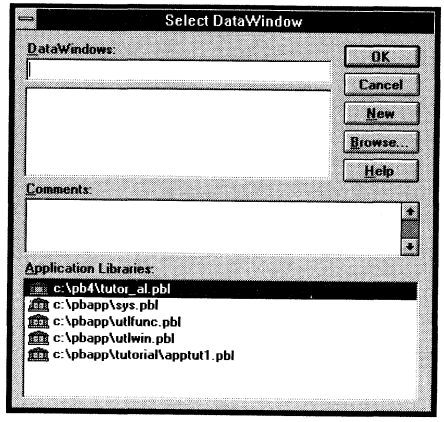
Now you will select a DataWindow object to display in the w_tut_report window.

1 Make sure you are in the Window painter with the w_tut_report window displayed.
If you are not, open the Window painter and select the *w_tut_report* window.



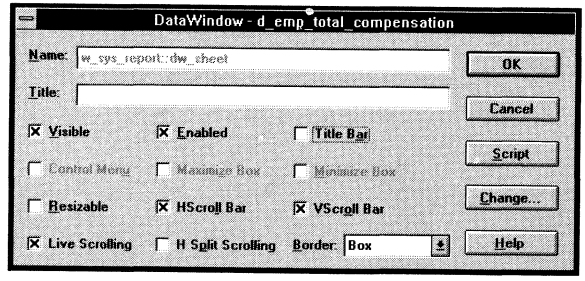
2 Double-click inside the DataWindow control.

The Select DataWindow dialog box displays.



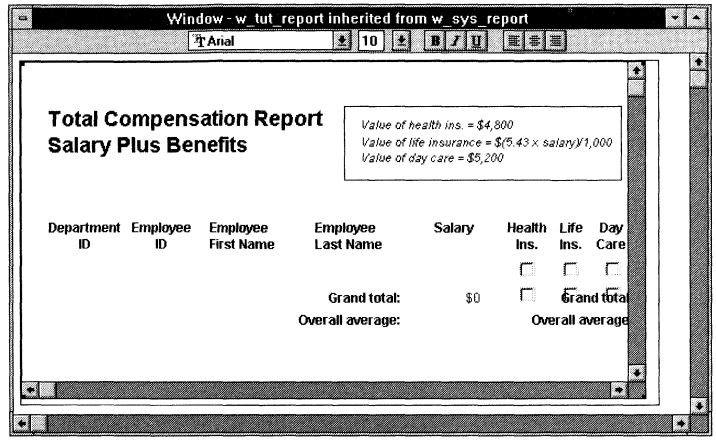
3 Click the line that ends with *apptut1.pbl* in the Application Libraries box. Double-click *d_emp_total_compensation*.

The DataWindow dialog box displays some of the attributes of the dw_sheet DataWindow control.



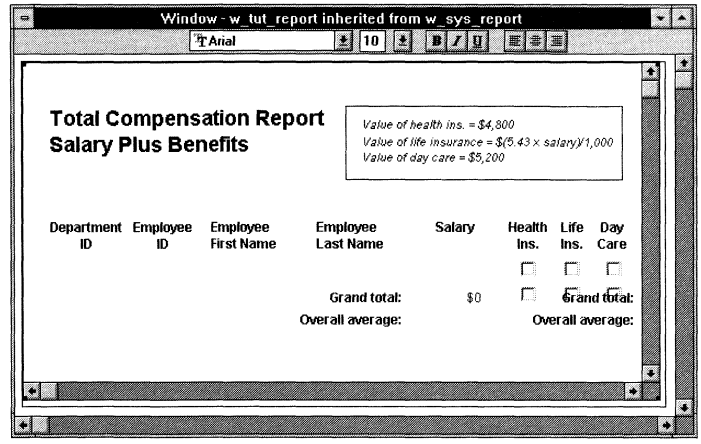
4 Click *OK*.

The Window painter workspace displays.



5 Adjust the size of the DataWindow control if necessary.

If you do not see all the columns in the DataWindow control, make the control wider, enlarging the window, if possible.



6 Select File➤Save from the menu bar.

PowerBuilder saves your changes.

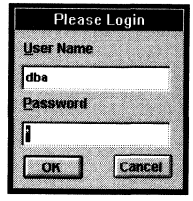
Run the application

Where you are
Lesson 8 Associating DataWindow Objects with DataWindow Controls
Select DataWindow objects for w_tut_shared
Add scripts for the dw_sheet DataWindow control
Select DataWindow object for w_tut_report
➡ Run the application

Now you will verify that the DataWindows work properly by running the Application Library tutorial.

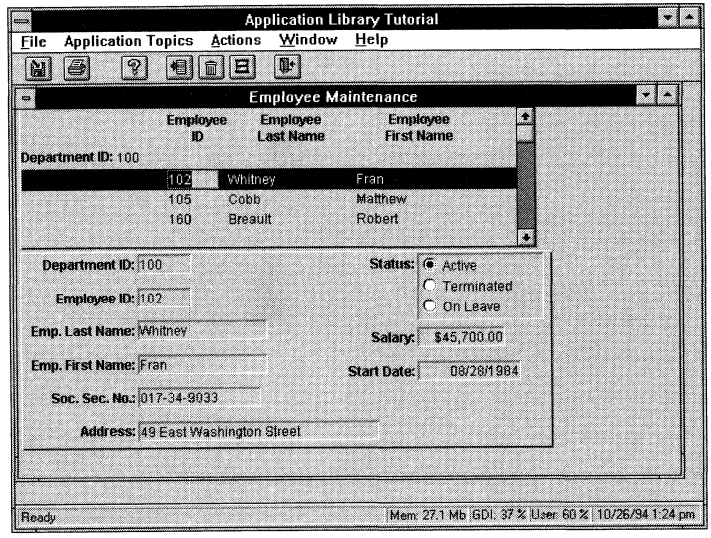
1 **Click the *Run* button in the PowerBar**
or
Select File➤Run from the menu bar.
If PowerBuilder prompts you to save changes, click *Yes*.

The frame window's Open event calls the f_login function, which displays the w_login dialog box.



2 Type *sql* (or whatever password you specified in the tutor_al.ini file). Click *OK*.

The application connects to the database and displays the Employee Maintenance window, populating the DataWindow controls with rows retrieved from the Powersoft Demo database.

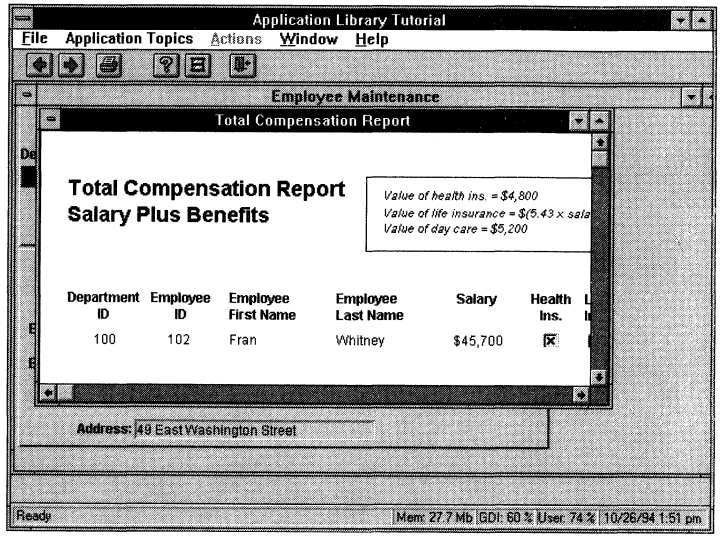


3 Click rows in the top DataWindow control and verify that detail information displays in the bottom DataWindow control.
Add an employee.
Delete an employee.
Modify employee information.
Select File➤Save to update the database.

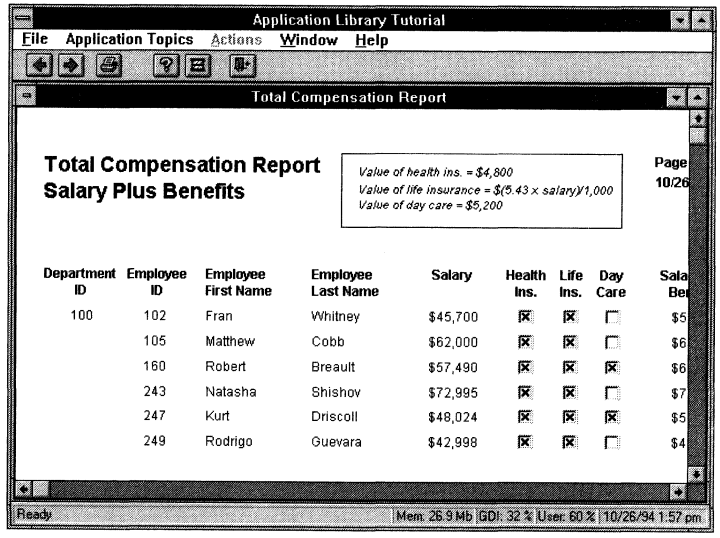
> **If you cannot save data**
> If error messages are displayed when trying to update the database, the wf_log window function may have been defined incorrectly.

4 Select Application Topics➤Total Compensation Report from the menu bar.

The Total Compensation Report window displays.



5 Display the full window by selecting Window➤Layer from the menu bar.



6 Use the scrollbars to browse the report.

7 Exit the application by pressing ALT+F4.

If your application has problems

Your application should run without errors. If an error occurs, double-check your scripts to make sure they were entered correctly.

If problems persist, use Debug to help solve them.

For information on Debug, see the PowerBuilder *User's Guide.*

What to do next

This completes the Application Library tutorial. To continue learning about the Application Library and PowerBuilder application development, you can:

- **Review this tutorial** to reinforce the basic application development techniques.
- **Read the rest of this manual** to understand the functions and objects available to you.
- **Review the Time Management and Pubs sample applications** to see how to use the Application Library to create a robust MDI application.
- **Read the *Building Applications* manual** for additional design, development, and deployment information.
- **Enhance the tutorial application** by adding functionality provided by other types of application framework windows.

PART THREE

Object Reference

This part describes the objects in the Application Library.

CHAPTER 4

Window Objects

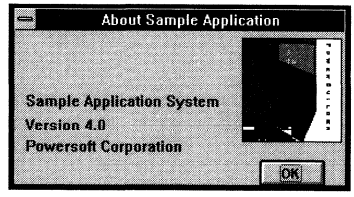
About this chapter

This chapter describes the window objects in the Application Library. It includes a description and sample of each window, as well as information on each window's controls, events, and functions. Windows are listed in alphabetical order.

w_about

Description

Displays the application name and version number.



Type

Response

Library

UTLWIN.PBL

Invocation

Open (**w_about**)

Controls used

Control	Control type
cb_ok	CommandButton

Control events

Control	Event	Description
cb_ok	Clicked	Closes the w_about window.

Usage

Open this window or a customized descendant from your application's Help➤About menu item.

Example

This example opens window w_about in the Clicked event of the menu item Help➤About in the main menu.

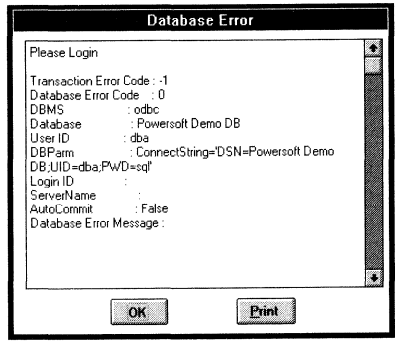
```
Open (w_about)
```

w_db_error

Description

Displays the current database error that is passed from the global function f_db_error. It enables the user to print the error message if desired, and then continue with the application session.

The f_db_error global function opens the window.



Type

Response

Library

UTLWIN.PBL

Invocation

f_db_error (*transactionobject, errormessage*)

Parameter	Description
transactionobject	Name of the programmer-specified transaction object currently being used
errormessage	String containing the user-defined error message

Controls used

Control	Control type
cb_ok	CommandButton
cb_print	CommandButton
mle_message	MultiLineEdit

Control events

Control	Event	Description
cb_ok	Clicked	Closes the w_db_error window
cb_print	Clicked	Creates a print file using the database error message information and prints the file

Usage

You might use this window by coding the f_db_error function in a script for a DataWindow control's DBError event.

Example

This example calls global function f_db_error and passes the default transaction object SQLCA and an application-specific error message. This call opens window w_db_error and displays the application message along with the specific database error message.

```
f_db_error (SQLCA,    &
   "Database error for Test Application")
```

See also

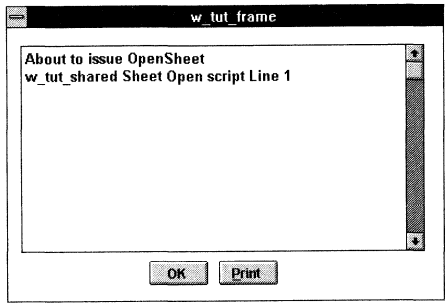
f_db_error
f_debug_box
f_error_box
f_get_string
w_debug_box
w_error_box
w_get_string

w_debug_box

Description

Displays an error message passed from global function f_debug_box. This window object acts similarly to the window displayed using the PowerBuilder MessageBox function; however, this window is non-modal (it does not require immediate response and can be left open).

The f_debug_box global function opens this window.



Type

Popup

Library

UTLWIN.PBL

Invocation

f_debug_box (*windowtitle, errormessage*)

Parameter	Description
windowtitle	String specifying w_debug_box window title. Usually contains the title of the window in which the error occurred.
errormessage	String containing the error message to be displayed.

Controls used

Control	Control type
cb_ok	CommandButton
cb_print	CommandButton

Control events

Control	Event	Description
cb_ok	Clicked	Closes the w_debug_box window
cb_print	Clicked	Prints the displayed message

Usage

Use this window from any event when building and debugging applications.

Example

This example opens window w_debug_box in the Clicked event of an OK CommandButton and checks the input variables (SingleLine Edit text values) before passing these variables as parameters.

```
f_debug_box("Username entered "+ sle_username.text)
f_debug_box("Password entered "+ sle_password.text)
```

See also

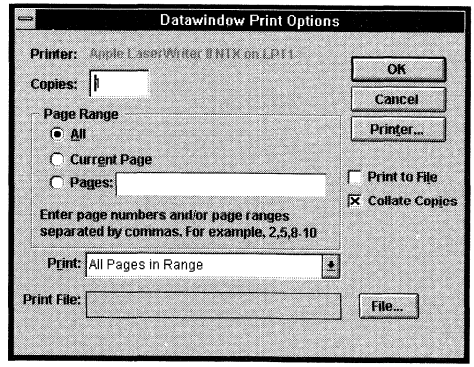
f_db_error
f_debug_box
f_error_box
f_get_string
w_db_error
w_error_box
w_get_string

w_dw_print_options

Description

Provides a Print Options dialog box for a DataWindow control, allowing you to control which pages are printed and the print destination.

The f_dw_print global function opens this window and prints the DataWindow.



Type Response

Library UTLWIN.PBL

Invocation **f_dw_print** (*dwcontrol, numberofcopies, filename*)

Parameter	Description
dwcontrol	DataWindow variable of the DataWindow control whose contents will be printed
numberofcopies	Integer specifying the number of copies to be printed
filename	String containing the fully qualified name of the file to contain the DataWindow contents

Controls used

Control	Control type
cb_cancel	CommandButton
cb_file	CommandButton
cb_ok	CommandButton
cb_printer	CommandButton
cbx_collate	CheckBox
cbx_print_to_file	CheckBox
ddlb_range	DropDownListBox
em_copies	EditMask
gb_1	GroupBox
rb_all	RadioButton
rb_current_page	RadioButton
rb_pages	RadioButton
sle_page_range	SingleLineEdit
sle_printer	SingleLineEdit
sle_printer	SingleLineEdit

Control events

Control	Event	Description
cb_cancel	Clicked	Closes the window, returning *-1* in Message.DoubleParm
cb_file	Clicked	Displays the Print to File dialog box, allowing the user to specify a filename
cb_ok	Clicked	Closes the window, returning *1* in Message.DoubleParm
cb_printer	Clicked	Displays the Windows Printer Setup dialog box, allowing the user to change the default printer
rb_all	Clicked	Specifies that all pages will be printed
rb_current_page	Clicked	Specifies that the current page will be printed
rb_pages	Clicked	Specifies that the pages named in the sle_page_range field will be printed

Window events

Event	Description
Open	Initializes fields and variables using the passed parameter

Window functions

wf_disable_printfile Hides all items related to printing a file.

- Parameters: None.
- Returns: None.

wf_enable_printfile Shows all items related to printing a file.

- Parameters: None.
- Returns: None.

wf_page_range Sets fields and variables, depending on which RadioButton is clicked.

- Parameters: RadioButton specifying which page range option is clicked.
- Returns: None.

Instance variables

Variable	Data type	Access
idw_dw	DataWindow	Public
is_page_range	String	Public

Usage

Use this window to allow users to control the printing of DataWindow pages.

If you use this window apart from the f_dw_print function:

- Before opening the window, ensure that datawindow.print.copies is set to zero and datawindow.print.file is set to "" (an empty string). Use the Describe function to query these values and the Modify function to update them.
- Remember that this window does not actually print the DataWindow; to print the DataWindow, issue a Print function after the window closes.
- After the window closes, check Message.DoubleParm before printing the DataWindow. A *1* indicates that the user made print specifications and clicked OK; *-1* indicates that the user clicked Cancel.

Example

This example uses the f_dw_print function to open the w_dw_print_options window.

```
string   ls_filename = ""
integer  li_num_copies = 0
integer  li_return

li_return = f_dw_print (dw_sheet, &
   li_num_copies, ls_filename)
IF li_return = -1 THEN
   MessageBox("Print Error", &
      "User pressed Cancel or an error occured")
END IF
```

See also

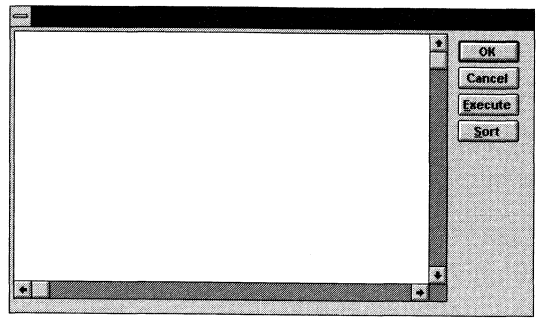
f_dw_print
f_print_file

w_dw_select

Description

Provides a reusable selection list window for DataWindows. The window is populated using a SQL statement you specify. The user then picks one of the selected rows to be passed back to the calling window.

The f_select_data global function opens this window.



Type

Response

Library

UTLWIN.PBL

Invocation

f_select_data (*sql, title, columns, dwcontrol, row, autoquery*)

Parameter	Description
sql	String containing SQL statement used to build a DataWindow.
title	String specifying title for w_dw_select window.
columns	String specifying column mapping. Composed of one or more pairs of column names from the calling DataWindow equated to a column number in the passed SQL statement. This mapping defines which columns in the calling DataWindow will be updated.
dw	DataWindow variable identifying the DataWindow control that is to be updated when the user selects a row in the selection window.
row	Long indicating the row of the calling DataWindow that is to be updated when the user selects a row in the selection window.

Parameter	Description
autoquery	Boolean indicating whether to automatically retrieve the data in the selection window (TRUE) or allow the user to enter retrieval criteria (FALSE).

Tip
The window's Open event processes these parameters within the str_select_parms structure.

Controls used

Control	Control type
dw_created	DataWindow
cb_ok	CommandButton
cb_cancel	CommandButton
cb_sort	CommandButton
cb_query	CommandButton

Control events

Control	Event	Description
cb_ok	Clicked	Triggers the DoubleClicked event in the dw_created DataWindow control.
cb_cancel	Clicked	Closes the window without performing any updates to the calling DataWindow.
cb_sort	Clicked	Opens the w_sort window to allow the user to specify a sort order for the dw_created DataWindow control.
cb_query	Clicked	Toggles the DataWindow in and out of query_mode. If the window was in query_mode, then a Retrieve for dw_created is also performed. The text of the button changes from *Query* to *Execute Query* as the mode changes.

Control	Event	Description
dw_created	DoubleClicked	Indicates that the user has selected a row and that the data in the columns requested are set on the calling DataWindow. This event assumes that the data types of the source and target columns are the same. It also closes the window.
dw_created	RetrieveEnd	Enables the cb_ok CommandButton if rowcount > 0.
dw_created	RowFocusChanged	Highlights the current row.

Window events

Event	Description
Open	Transfers the parameters passed on the message object to local variables. It then creates the DataWindow from the SQL statement, sizes the DataWindow and window to fit the created DataWindow object, positions the CommandButtons on the window, and issues a PostEvent to the post_open event. This PostEvent is issued so that the window displays before the actual retrieve, if there is one.
post_open (user event)	Checks the *autoquery* parameter, and if it is TRUE it causes the DataWindow to be retrieved automatically.

Instance variables

Variable	Data type	Access
parm	str_select_parms	Public

Usage

Use this window to display a list of items from which the user can select one item to be returned to the calling window. The user selects an item either by double-clicking it, or by single-clicking and then clicking OK.

Example

This example calls the f_select_data global function from a DataWindow control (typically from the Clicked or DoubleClicked event). This call opens window w_dw_select, displaying rows retrieved by the specified SQL statement.

```
string col_name
long row

col_name = GetObjectAtPointer()
col_name = f_get_token(col_name,"~t")
```

```
IF col_name = 'title_id' THEN
   row = GetClickedRow()
   f_select_data ("SELECT title_id,title, price,&
      ytd_sales, type FROM titles",  &
      "Select Title", "title_id=1,title=2,price=3, &
      ytd_sales=4, type=5", this, row, false )
END IF
```

See also

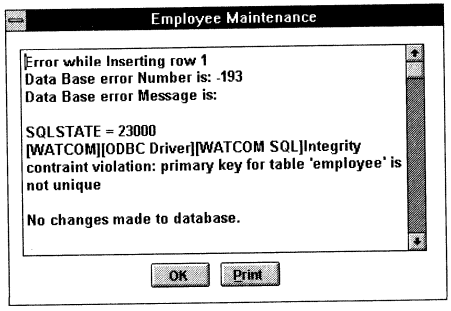
f_get_token
f_select_data
str_select_parms
w_select

w_error_box

Description

Displays an error message passed from global function f_error_box. This window acts similarly to the PowerBuilder MessageBox function; however, this window is non-modal (it does not require immediate response and can be left open).

The f_error_box global function opens this window.



Type

Response

Library

UTLWIN.PBL

Invocation

f_error_box (*windowtitle*, *errormessage*)

Parameter	Description
windowtitle	String containing w_error_box window title. Usually contains the title of the window in which the error occurred.
errormessage	String containing the error message to be displayed.

Controls used

Control	Control type
cb_ok	CommandButton
cb_print	CommandButton

Control events

Control	Event	Description
cb_ok	Clicked	Closes the window
cb_print	Clicked	Prints the message displayed

Usage

You typically use this window by coding the f_error_box function in a script for a DataWindow control's DBError event.

Example

This example opens window w_error_box in the Clicked event of a Print CommandButton if the PrintOpen function fails.

```
integer li_x

li_x =  PrintOpen( )
IF li_x = -1 THEN
   f_error_box(parent.title,  &
      "Error in Clicked event of object cb_print")
END IF
```

See also

f_db_error
f_debug_box
f_error_box
f_get_string
w_db_error
w_debug_box
w_get_string

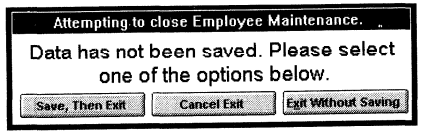
w_exit_status

Description

Displays a message if the current window is closing and data has been modified but not saved or if changed data is about to be lost after moving to a different row or window.

The f_exit_status global function opens this window and prompts the user to choose one of the following options:

- **Save, Then Exit** Save data before continuing
- **Exit Without Saving** Do not save data before continuing
- **Cancel Exit** Cancel the close or move and redisplay the window



Type

Response

Library

UTLWIN.PBL

Invocation

f_exit_status (*windowtitle*, *saveormove*)

Parameter	Description
windowtitle	String specifying w_exit_status window title. Usually contains the title of the window to be closed.
saveormove	String indicating whether data will be lost due to closing a window or moving from a DataWindow row in which data has been changed for the row: ♦ **S** – Unsaved changes will be lost because the window is closing ♦ **M** – Unsaved changes will be lost because the user is moving from a DataWindow row with unsaved changes

Controls used

Control	Control type
cb_save	CommandButton
cb_cancel	CommandButton

Control	Control type
cb_exit	CommandButton
st_heading	StaticText

Control events

Control	Event	Description
cb_save	Clicked	Returns an S and closes the w_exit_status window
cb_cancel	Clicked	Returns a C and closes the w_exit_status window
cb_exit	Clicked	Returns an E and closes the w_exit_status window

Window events

Event	Description
Open	Uses the *windowtitle* parameter to set the window.title of window w_exit_status

Usage

In a window's CloseQuery event, check to see if any changed data has not been saved to the database. If there is unsaved data, use the f_exit_status function to open the w_exit_status window.

Example

This example checks to see if there is any unsaved data in the window. If TRUE, use f_exit_status to open w_exit_status and ask the user if information should be saved.

```
string ls_status

IF dw_sheet.ModifiedCount + &
   dw_sheet.DeletedCount <> 0 THEN
   ls_status = f_exit_status(this.title)
END IF

CHOOSE CASE ls_status
   CASE "S"
      // Save data then exit.
   CASE "E"
      // Exit the application.
   CASE "C"
      // Cancel the exit.
END CHOOSE
```

See also

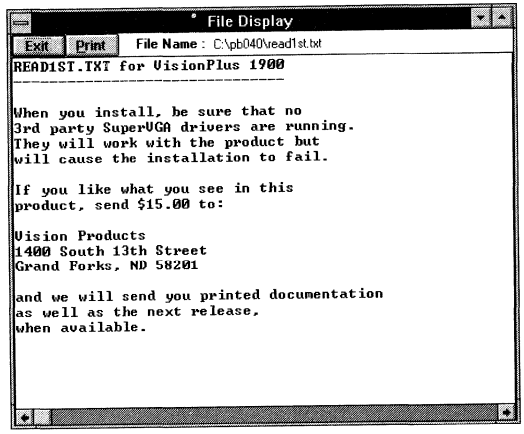
f_exit_status

w_file_display

Description

Displays a file with the TXT extension and allows users to print the displayed information.

The f_display_file global function opens this window.



Type

Popup

Library

UTLWIN.PBL

Invocation

f_display_file (*filename*)

Parameter	Description
filename	String containing the fully qualified name of the file to be displayed in the w_file_display window. This file must use the TXT extension.

Controls used

Control	Control type
cb_exit	CommandButton
cb_print	CommandButton
dw_displayfile	DataWindow
st_filename	StaticText
st_name	StaticText

Control events

Control	Event	Description
cb_exit	Clicked	Closes the w_file_display window
cb_print	Clicked	Prints the displayed information

Window events

Event	Description
Open	Imports *filename* into the dw_displayfile DataWindow control
Resize	Resizes the DataWindow based on the size of the window

Usage

Use the f_display_file function and w_file_display window to display text files from within your application.

Example

This example displays file C:\PB\CUSTOMER.TXT.

```
string ls_filename

ls_filename = "c:\pb\customer.txt"
f_display_file(ls_filename)
```

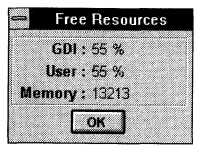
See also

f_display_file
f_print_file

w_get_free_resources

Description

Displays the free system resources at the time it opened. These resources consist of: graphic display interface (GDI), user, and free memory (which includes virtual memory).



Type

Response

Library UTLWIN.PBL

Invocation Open (**w_get_free_resources**)

Controls used

Control	Control type
cb_ok	CommandButton

Control events

Control	Event	Description
cb_ok	Clicked	Closes the w_get_free_resources window

Window events

Event	Description
Open	Sets the static text with the current resource values.

Usage

Use this window to allow your application to display free system resources. You could add a Help menu item that displays this window.

This window uses the GetFreeSystemResources and GetFreeSpace Windows functions, which are declared as local external functions:

```
function uint GetFreeSystemResources(uint
SysResource) library 'user.dll'

function long GetFreeSpace(uint SysResource) library
'kernel.dll'
```

Example

This example opens the w_get_free_resources window object.

```
Open (w_get_free_resources)
```

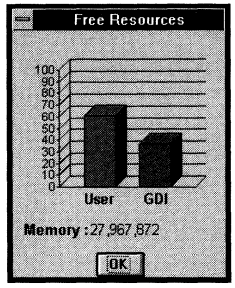
See also

m_sys_frame
w_get_free_resources_graph

w_get_free_resources_graph

Description

Displays the free system resources as a graph at the time it is opened. These resources consist of: graphic display interface (GDI), user, and free memory (which includes virtual memory).



Type

Response

Library

UTLWIN.PBL

Invocation

Open (**w_get_free_resources_graph**)

Controls used

Control	Control type
gr_1	Graph
cb_ok	CommandButton

Window events

Event	Description
Open	Sets the static text and graph control with the current resource values.

Control events

Control	Event	Description
cb_ok	Clicked	Closes the window

Usage

Use this window to allow your application to display a graph of free system resources. You might consider adding a Help menu item that displays this window.

This window uses the GetFreeSystemResources and GetFreeSpace Windows functions, which are declared as local external functions:

```
function uint GetFreeSystemResources(uint
SysResource) library 'user.dll'

function long GetFreeSpace(uint SysResource) library
'kernel.dll'
```

Example

This example opens the w_get_free_resources_graph window object.

```
Open (w_get_free_resources_graph)
```

See also

m_sys_frame
w_get_free_resources

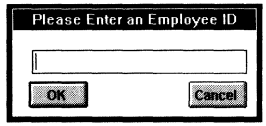
w_get_string

Description

Displays a window that prompts the user to enter a string. The window is passed parameters that determine the maximum size of the string the user will input, the character case, and the current window title.

This window resizes automatically based on the maximum length of the string the user will input.

The f_get_string global function opens this window.



Type

Response

Library

UTLWIN.PBL

Invocation

f_get_string (*windowtitle, maxlength, caseindicator, currentvalue*)

Parameter	Description
windowtitle	String indicating the title to be used in w_get_string.

Parameter	Description
maxlength	Integer specifying the maximum number of characters that can be entered.
caseindicator	Single-character string indicating the case of the string to be entered: U – Uppercase, L – Lowercase, A – Any case.
currentvalue	Optional string containing the current value of what is to be modified.

Controls used

Control	Control type
cb_ok	CommandButton
cb_cancel	CommandButton

Control events

Control	Event	Description
cb_ok	Clicked	Returns the string entered by the user and closes the w_get_string window
cb_cancel	Clicked	Returns a NULL string and closes the w_get_string window

Window events

Event	Description
Open	Reads the input parameters and centers the window on its parent

Instance variables

Variable	Data Type	Access
ii_max_len	Integer	Public

Usage

Use this window to prompt users for key variables or other additional information.

Examples

This example prompts the user for a string that is up to 40 characters long and uppercase only. The title of w_get_string will be equal to the title of the calling window.

```
string   ls_string
ls_string = f_get_string(parent.title,40,"U")
```

This example prompts the user for multiple strings different in length and case type. The title of the new window opened is equal to the information being requested.

```
string   ls_custnum, ls_dept, ls_custname
ls_custnume = f_get_string("Customer number",8,"U")
ls_dept     = f_get_string("Department", 4, "U")
ls_custname = f_get_string("Customer name", 40, "A")
```

See also

f_get_string
str_parms

w_hold_parms

Description

Contains the functions that are used to manipulate values in a parameter stack. You can use this window to replace the use of global variables.

This window, which is invisible by default, must be opened during the Open event of the application.

Type

Popup

Library

UTLFUNC.PBL

Invocation

Open (**w_hold_parms**)

Window functions

wf_add_item Adds or updates the string item in a DataWindow to the string value. It does not return any parameters.

wf_get_item Finds the item name that corresponds to the passed parameter value. It returns the string value of the item found. If nothing is found, a NULL string is returned.

wf_pop_item Removes the first value in the stack array. It returns the value to the calling routine.

wf_push_item Adds an item to the parameter stack. It also increments the stack array by one.

Instance variables

Variable	Data type	Access
is_stack[ ]	String	Public
ii_num_in_stack	Integer	Public

Usage

The main purpose of this window (to pass parameters when opening and closing windows) has been replaced with the functionality provided by OpenWithParm and CloseWithReturn functions. Window object w_hold_parms has been retained for backward compatibility and for the capabilities that wf_add_item and wf_get_item provide.

Example

This example shows the usage of wf_pop_item and wf_push_item functions:

```
// In the event called to open window w_foo
// push the value needed to retrieve the data in
// window w_foo onto the stack.
w_hold_parms.wf_push_item(ret_arg)
Open(w_foo)


// In the open event of w_foo get the retrieve
// argument from the stack.
string ret_arg
ret_arg = w_hold_parms.wf_pop_item()
dw_1.retrieve(ret_arg)
```

This example shows how to set a globally accessible string named foo into the list:

```
// Set a variable name foo into the list.
w_hold_parms.wf_add_item('foo',value)


// In some other event, access the value of foo.
string value
value = w_hold_parms.wf_get_item('foo')
```

See also

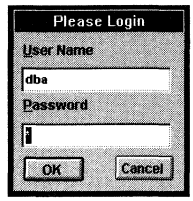
f_pop_parm
f_push_parm
f_set_parm

w_login

Description

Displays a window that prompts the user for a user name and a password to connect to the database of the current application.

The f_login global function opens this window.



Type

Response

Library

UTLWIN.PBL

Invocation

f_login (*inifilename*)

Parameter	Description
inifilename	The path and complete name of the INI file name associated with the current application. The INI file contains specific DBMS information used for connecting to the database.

Controls used

Control	Control type
st_1	StaticText
st_2	StaticText
sle_username	SingleLineEdit
sle_password	SingleLineEdit
cb_ok	CommandButton
cb_cancel	CommandButton

Control events

Control	Event	Description
cb_ok	Clicked	Ensures that the user entered a user name and a password. Depending on the DBMS being used, the event will set either the user id, logid, dbpass, or logpass and attempt to connect to the database. If the connection is unsuccessful, the event will display the specific DBMS error message and allow the user three login attempts. It passes a boolean parameter back to the calling function indicating the success of the login.
cb_cancel	Clicked	Passes a boolean parameter back to the calling function indicating an unsuccessful login, and closes the window.
st_1	Clicked	Sets focus to sle_username.
st_2	Clicked	Sets focus to sle_password.

Window events

Event	Description
Open	Sets the default transaction object to corresponding information in the application's INI file.

Instance variables

Variable	Data type	Access
ii_attempts	Integer	Public
ib_connected	Boolean	Public

Usage

You typically use this window in the initial window of an application to prompt users to log in to the database. For example, the post_open user event of the w_sys_frame application framework window uses the f_login function to open this window.

The user name and password are not automatically converted to upper or lowercase. You can change this by setting the case attribute on the SingleLineEdits.

Example

This example call the f_login function in the Open event of the frame window. If the user is unable to log in, the application closes.

```
IF NOT f_login("c:\pubs\pubs.INI") THEN
   halt close
END IF
```

See also

f_db_error
f_login

w_mdi_clock

Description

Displays date, time, memory information, and other user-specified text in the lower-right corner of an MDI frame window. You must call a function to reposition w_mdi_clock when the MDI frame is moved or resized.

Type

Popup

Library

UTLWIN.PBL

Invocation

OpenWithParm (**w_mdi_clock** *bitmask*)

Parameter	Description
bitmask	Bit mask that specifies which standard items to display: ♦ 1 Time and date ♦ 2 User memory ♦ 4 GDI memory ♦ 8 Free memory

Controls used

Control	Control type
st_time	StaticText

Window events

Event	Description
Open	Determines display information and sets the timer process, which runs at one-minute intervals.
Close	Removes user object items from w_mdi_clock.

Event	Description
Timer	Refreshes window items.

Window functions

wf_add_item Adds an item to w_mdi_clock.

- Parameters:
  - Integer specifying item width.
  - Alignment enumerated variable specifying justification (center!, left!, or right!).
  - Single-character string used to size the four predefined items (T=time and date, G=GDI memory, U=user memory, or M=free memory). This is an empty string for user-defined items.
- Returns: Integer indicating the number of items displayed in w_mdi_clock.

wf_calc_sizes Calculates the total width of the w_mdi_clock window.

- Parameters: None
- Returns: None

wf_del_item Removes the specified item from w_mdi_clock.

- Parameters: Integer specifying the position of the item to be deleted.
- Returns: Boolean. Returns TRUE if the item was deleted successfully and FALSE if it was not.

wf_num_items Returns the number of items currently displayed in w_mdi_clock.

- Parameters: None
- Returns: Integer specifying the number of items currently displayed in w_mdi_clock.

wf_parent_resized Moves the window so that it is positioned properly on the MicroHelp bar of the MDI frame.

- Parameters: None
- Returns: None

wf_predefined_action Indicates whether the passed action is one of the four predefined actions (T, G, U, and M).

- Parameters: Single-character string specifying the action.
- Returns: Boolean. Returns TRUE if the passed action is one of the four predefined actions and FALSE if it is not.

wf_set_text Sets the contents of the specified w_mdi_clock position.

- Parameters:
  - Integer specifying the position to contain the new text
  - String containing the text
- Returns: Boolean. Returns TRUE if the text was set successfully and FALSE if it was not.

Instance variables

Variable	Default	Data type	Access
iw_parent_window	None	Window	Private
ii_menu_ht	0 (zero)	Integer	Private
ii_menu_ht2	74	Integer	Private
ii_not_menu_ht	0 (zero)	Integer	Private
ii_resizeable_offset	None	Integer	Private
ii_border	None	Integer	Private
ii_border_width	None	Integer	Private
ii_border_height	None	Integer	Private
item_cnt	0 (zero)	Integer	Private
u_mdi_clock_item[]	None	UserObject	Private
items_actions	None	String	Private
ib_init_phase	None	Boolean	Private

Usage

Use this window to display date, time, Windows memory information, and application-specific information in your MDI application's status bar.

You must add a custom user event to the MDI frame to reposition w_mdi_clock when the frame is moved. To do this, declare a Move user event, which uses the Windows message pbm_move. Call the wf_parent_resized function in this event. You must also add a call to wf_parent_resized in the MDI Frame window's Resize event.

You can open w_mdi_clock displaying four predefined items. From right to left, they are: T (time). G (GDI memory), U (user memory), and M (free memory). You can also use the wf_add_items and wf_set_text functions to add your own items to the left side of the display.

Tip
To examine an implementation of w_mdi_clock, see the Open, Resize, and Move events in the application framework window w_sys_frame.

Example

This example opens window w_mdi_clock displaying the four default items.

```
OpenWithParm(w_mdi_clock 1+2+4+8)
```

This example opens window w_mdi_clock displaying the time only.

```
OpenWithParm(w_mdi_clock 1)
```

Code this script in the Resize and Move events of the MDI frame.

```
IF IsValid(w_mdi_clock) THEN
   w_mdi_clock.wf_parent_resized( )
END IF
```

Use this code to add an item to w_mdi_clock.

```
integer  li_position

li_position =
w_mdi_clock.wf_add_item(350,center!,"")
w_mdi_clock.wf_set_text(li_position,"New item text")
```

Use this code to remove the leftmost item from w_mdi_clock.

```
integer  li_del_item

li_del_item = w_mdi_clock.wf_num_items( )
w_mdi_clock.wf_del_item(li_del_item)
```

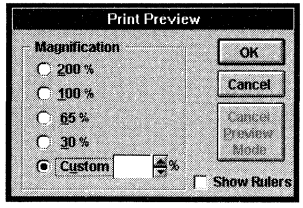
See also

u_mdi_clock_item
w_sys_frame

w_printzoom

Description

Controls DataWindow print preview.



Type

Response

Library

UTLWIN.PBL

Invocation

OpenWithParm (**w_printzoom**, *dwname*)

Parameter	Description
dwname	DataWindow that is being previewed

Controls used

Control	Control type
cbx_rulers	CheckBox
cb_cancel	CommandButton
cb_ok	CommandButton
em_custom	EditMask (spin control)
pb_cancel	PictureButton
rb_custom rb_30 rb_65 rb_100 rb_200	RadioButtons

Control events

Control	Event	Description
cb_cancel	Clicked	Closes the window

Control	Event	Description
cb_ok	Clicked	Sets the DataWindow to the state specified by the current control settings
em_custom	Spun (pbm_enchange)	Sets rb_custom.clicked to TRUE
pb_cancel_previe w	Clicked	Turns print preview off for the DataWindow and close the w_printzoom window
rb_30	Clicked	Sets the zoom percentage to 30%
rb_65	Clicked	Sets the zoom percentage to 65%
rb_100	Clicked	Sets the zoom percentage to 100%
rb_200	Clicked	Sets the zoom percentage to 200%
rb_custom	Clicked	Sets the zoom percentage to the spin control (em_custom)

Window events

Event	Description
Open	Reads the current settings of the passed DataWindow and sets the controls as required

Instance variables

Variable	Data type	Access
idw_dw	DataWindow	Public

Usage

The user can use this window:

- Toggle a DataWindow in and out of print preview mode
- Select the percentage of print zoom
- Display rulers when a DataWindow is in print preview mode

Example

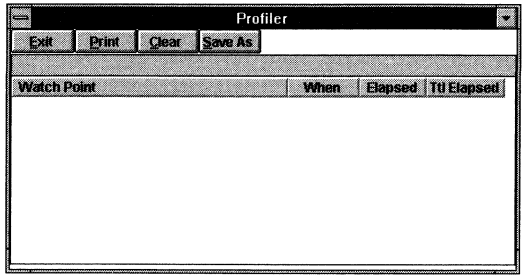
This example opens w_printzoom.

```
OpenWithParm (w_printzoom,dw_sheet)
```

w_profile

Description

Allows you to capture performance statistics on application windows while they are processing.



Type

Popup

Library

UTLWIN.PBL

Invocation

Open (**w_profile**)

Controls used

Control	Control type
cb_exit	CommandButton
cb_saveas	CommandButton
cb_print	CommandButton
cb_close	CommandButton
dw_1	DataWindow

Window events

Event	Description
Resize	Resizes the DataWindow to fit the window

Window functions

wf_checkpoint Finds the difference between the current time and the last wf_checkpoint (or wf_start). Also displays elapsed time and a message.

- *Parameters*: String containing message to be displayed with checkpoint time.
- *Return value*: None

wf_start Resets the DataWindow title with the parameter passed to it and resets the instance variables il_start_time and il_last_time.

- *Parameters*: String containing text to be displayed on title of dw_1 DataWindow control.
- *Return value*: None

Instance variables

Variable	Data type	Access
il_start_time	Long	Private
il_last_time	Long	Private

Usage

Use this window's wf_checkpoint function to write periodic messages to w_profile, allowing you to determine the length of interim processing.

Example

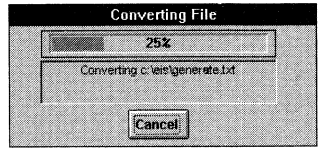
This example uses w_profile.

```
Open(w_profile)

// Set the title and reset the counters.
w_profile.wf_start("tracking calculation progress"

// Set checkpoints to determine length of
// phase 1 and phase 2.
w_profile.wf_checkpoint("Before calculation")
calc_project_phase1()
w_profile.wf_checkpoint("after Phase 1")
calc_project_phase2()
w_profile.wf_checkpoint("after Phase 2")
```

w_progress

Description

Displays a percentage complete indicator. It is a shaded bar showing the percentage of the process that is complete, a status area (text), and a Cancel button.



Type

Popup

Library

UTLWIN.PBL

Invocation

OpenWithParm (**w_progress**, *progressparms*)

Parameter	Description
progressparms	A structure of parameters used by w_progress. Use the str_progress structure to contain these parameters: ♦ Window that is to be notified if the user clicks the Cancel button or closes the Progress window ♦ Event to be triggered on the above window if canceled ♦ Title to be used on progress window

Controls used

Control	Control type
dw_progress	DataWindow
cb_cancel	CommandButton

Control events

Control	Event	Description
cb_cancel	Clicked	Triggers the passed event on the passed window and then closes the w_progress window

Window events

Event	Description
Open	Sets instance variables

Window functions

wf_progress Causes the displayed percentage and bar size to reflect the percentage passed as a parameter. You also use this function to display additional status information. Pass these parameters to wf_progress:

- *Parameters:* Real containing a value in the range 0 to 1.0 to indicate percent complete and an optional string that contains additional status information, such as the sub-task being performed.
- *Return value:* None

Instance variables

Variable	Data type	Access
ii_bar_width	Integer	Public
is_cancel_event	String	Public
iw_cancel_window	Window	Public

Usage

After opening the w_progress window, your long-running process calls wf_progress periodically, passing the percent complete. The user has the option to cancel the process at any time.

Example

This example opens the window w_progress:

```
str_progress parm
parm.title = 'Installing System'
parm.cancel_window = w_install_window
parm.cancel_event = "install_canceled"
OpenWithParm (w_progress,parm)
```

This example calls wf_progress:

```
w_progress.wf_progress(current_step/total_steps,&
   "Copying foo.pbl")
```

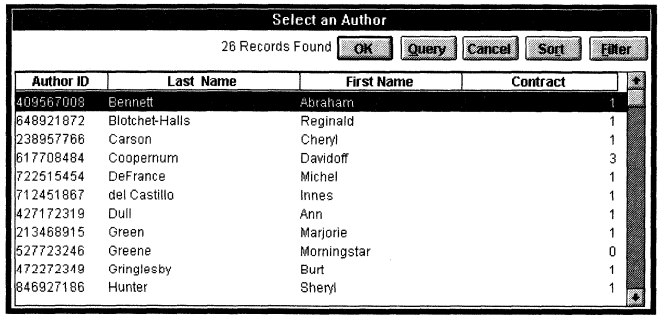
See also

str_progress

w_select

Description

Use this window as an ancestor for producing selection windows based on tables associated within an application (for example, searching for a particular customer ID number, a particular customer name, and so on.).



Type

Response

Library

UTLWIN.PBL

Invocation

OpenWithParm (*w_select_descendant, retrieveindicator*)

Parameter	Description
w_select_descendant	Window that is a descendant of w_select.
retrieveindicator	String value indicating whether the DataWindow should retrieve the data when the window is opened ("TRUE") or open in query mode ("FALSE").

Controls used

Control	Control type
cb_cancel	CommandButton
cb_filter	CommandButton
cb_search	CommandButton
cb_ok	CommandButton
cb_sort	CommandButton
dw_1	DataWindow
st_num	StaticText

Control events

Control	Event	Description
cb_ok	Clicked	Not used in the ancestor window. This event takes data from selected columns and passes it back to the calling process, closing the window at the same time. See the example below.
cb_sort	Clicked	Not used in the ancestor window. This event expects the descendant to call one of the sort routines. See the example below.
cb_filter	Clicked	Allows the user to enter information that will filter the data that is displayed.
cb_search	Clicked	Causes the DataWindow to perform a Retrieve using the query information that has been entered.
cb_cancel	Clicked	Closes the window and returns without passing any information back.

Window events

Event	Description
Open	Determines whether to do the retrieve automatically or wait until the user enters search criteria

Window functions

wf_check_row_count Takes the passed SQL statement and determines the number of rows that will be returned if the SQL statement is executed.

- *Parameters.* String containing SQL statement to be executed.
- *Return value.* Boolean. Returns TRUE if the count is over 500 and FALSE if it is not.

wf_set_buttons Sets the state of the command buttons to enabled or disabled as necessary. The scope of this function is private.

- *Parameters.* Boolean. TRUE enables buttons and FALSE disables them.
- *Return value.* None

Instance variables

Variable	Data type	Access
il_cur_row	Long	Protected

Variable	Data type	Access
ib_data_ok	Boolean	Protected
istr_parm	str_parms	Protected
il_row_count	Long	Private
ii_num-selected	Integer	Private
ii_columns	Integer	Private
ii_max_rows	Integer	Private
ib_query_mode	Boolean	Private

Usage

To use this window, you inherit (or copy an existing descendant) from this window, place a new DataWindow that displays the necessary data, and then code certain event scripts in the descendant.

Add a window title
The default title for this window is *Select a*. Customize the title based on the window's function.

Example

This example opens the w_select_desc window (descendent window) and retrieves all of the data.

```
OpenWithParm(w_select_desc,true)
```

This example is placed in the descendent window's cb_ok:clicked event script.

```
// Need to pass two strings back to calling objects
// so place them in the structure and return.

if not ib_data_ok then return
istr_parm.string_arg[1]=  &
   dw_1.GetItemString(il_cur_row, "au_id")
istr_parm.string_arg[2]= &
   dw_1.GetItemString(il_cur_row, "au_fname") &
   + " " + &
dw_1.GetItemString(il_cur_row,"au_lname"
CloseWithReturn(parent,istr_parm)
```

This is an example of what would be found in the Clicked event for the cb_sort control on a descendent window.

```
f_sort_order(dw_1, "Author ID:au_id, &
   Last Name:au_lname," &
   + "First Name:au_fname,Contract:contract")
```

See also

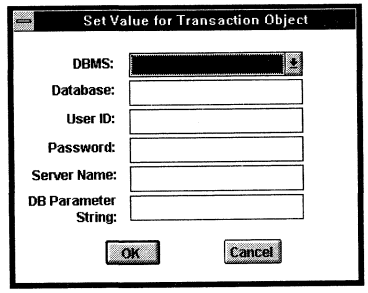
f_sort_order
str_parms

w_set_sqlca

Description

Used to enter database connection information if it is not provided in some other way.

The f_set_sqlca global function opens this window.



Type

Response

Library

UTLWIN.PBL

Invocation

f_set_sqlca ( )

Controls used

Control	Control type
sle_database	SingleLineEdit
sle_dbparm	SingleLineEdit
sle_dbpass	SingleLineEdit
sle_server	SingleLineEdit
sle_userid	SingleLineEdit
cb_close	CommandButton
cb_ok	CommandButton

Control events

Control	Event	Description
cb_ok	Clicked	Sets the transaction object values and attempts to connect to the database. If the connection succeeds, the window will close. If the connection fails, a database error message will display.
cb_cancel	Clicked	Closes the w_set_sqlca window.

Window events

Event	Description
Open	Displays existing transaction object information, if any

Usage

Use this window as an alternative method of establishing database connection information. This window is best used in the development and testing environment. Other methods (such as establishing an application INI file) are preferred in the production environment, since they do not require the user to enter database-specific information.

Example

This example opens the window w_set_sqlca.

```
f_set_sqlca( )
```

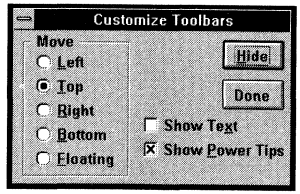
See also

f_set_sqlca

w_set_toolbars

Description

Allows the user to specify the position of the toolbar, whether the text for the toolbar buttons is displayed or not, and whether the toolbar is visible or not.



Type Response

Library UTLWIN.PBL

Invocation OpenWithParm (**w_set_toolbars**, *framewindow*)

Parameter	Description
framewindow	Window variable that contains the MDI frame window for the toolbar

Controls used

Control	Control type
cb_done	CommandButton
cb_visible	CommandButton
cbx_showtext	CheckBox
cbx_show_tips	CheckBox
rb_bottom	RadioButton
rb_floating	RadioButton
rb_left	RadioButton
rb_right	RadioButton
rb_top	RadioButton

Control events

Control	Event	Description
cb_done	Clicked	Closes the window
cb_visible	Clicked	Toggles the toolbar from visible to invisible
cbx_showtext	Clicked	Toggles the display of toolbar text
cbx_show_tips	Clicked	Toggles the display of PowerTips
rb_bottom	Clicked	Moves the toolbar to the bottom of the frame
rb_floating	Clicked	Changes the toolbar into a floating toolbar
rb_left	Clicked	Moves the toolbar to the left edge of the frame
rb_right	Clicked	Moves the toolbar to the right edge of the frame
rb_top	Clicked	Moves the toolbar to the top of the frame

Window events

Event	Description
Open	Sets instance variables and control status as required to reflect the current state of the toolbar for the active sheet or frame

Instance variables

Variable	Data type	Access
iw_win_ref	Window	Public
iapp_ref	Application	Public

Usage

Most PowerBuilder applications display the w_set_toolbars window through a Window➤Toolbars menu item.

> **Tip**
> Use w_set_toolbars to provide a way for users to redisplay a hidden toolbar.

Example

This example opens the w_set_toolbars window.

```
w_sys_frame frame

frame = mf_frame()
OpenWithParm(w_set_toolbars,frame)
```

> **Tip**
> The mf_frame function is an ancestor-level menu function that returns the MDI frame name.

See also

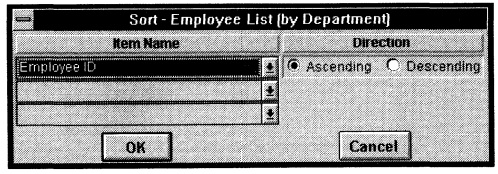
m_sys_frame
w_sys_frame

w_sort

Description

Allows the user to specify the sort order of a DataWindow. This window assumes that there will be a text object that corresponds to each visible column on the DataWindow and that the name of the text object is the same as the column name but with the _t suffix. If there is no matching text object, then the name of the column is displayed rather than the associated text.

DataWindow should not use grouping
Unpredictable results can occur if the DataWindow to be sorted contains grouping.



Type

Response

Library

UTLWIN.PBL

Invocation

OpenWithParm (**w_sort**, *sortparms*)

Parameter	Description
sortparms	A structure of parameters used by w_sort. Use the str_sort structure to contain these parameters: ♦ The DataWindow to be sorted ♦ The title to be placed on w_sort

Controls used

Control	Control type
cb_cancel	CommandButton
cb_ok	CommandButton
dw_sort	DataWindow
rb_bottom	RadioButton

Window events

Event	Description
Open	Sets instance variables and determines the columns on the DataWindow to be sorted. If the DataWindow is already sorted, it displays the current sort.

Control events

Control	Event	Description
cb_ok	Clicked	Sorts the passed DataWindow and closes the window
cb_cancel	Clicked	Closes the window without doing any sorting

Instance variables

Variable	Data type	Access
idw_dw	DataWindow	Public
is_title	String	Public

Usage

Use the w_sort window as a way of allowing users to control DataWindow sort order.

Example

This example opens the w_sort window.

```
str_sort parm
parm.dw = dw_sheet  // DataWindow to be sorted
parm.title = title  // Title for the w_sort window
OpenWithParm(w_sort,parm)
```

See also

f_sort_order
str_sort
w_sort_order

w_sort_order

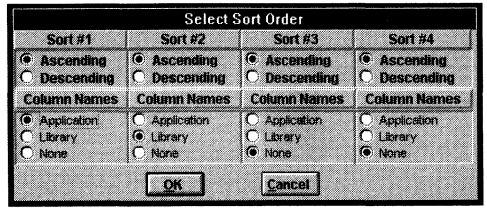
Description

Allows user control of DataWindow sort order. If the DataWindow is already sorted, the current sort order is displayed.

DataWindow should not use grouping
Unpredictable results can occur if the DataWindow to be sorted contains grouping.

The w_sort_order window allows the user to specify up to four levels of sorting for a DataWindow and it can display up to ten columns on which to sort.

The f_sort_order global function opens this window.



Type

Response

Library

UTLWIN.PBL

Invocation

f_sort_order (*dwname*, *sortcolumns*)

Parameter	Description
dwname	DataWindow control to be manipulated by w_sort_order.
sortcolumns	String containing the possible sort columns that the user can choose. The string list should be in the format of *Heading:column_name,Heading:column_name, Heading:column_name*, for up to ten columns.

Controls used

Control	Control type
cb_cancel	CommandButton
cb_ok	CommandButton
dw_sort	DataWindow

Control events

Control	Event	Description
cb_ok	Clicked	Finds the columns selected by the user in which to sort the parent window and returns the column list
cb_cancel	Clicked	Returns a NULL string and closes window w_sort_order

Window events

Event	Description
Open	Gets the sort order list from the message object and parses through the list in order to paste the column names on the DataWindow

Usage

Use the w_sort_order window as a way of allowing users to control DataWindow sort order.

Example

This example opens window w_sort_order allowing the user to choose columns in dw_1 (emp_name, dept_name, and emp_num) to sort in either ascending or descending order. The user can choose multiple sort columns.

```
f_sort_order(dw_1,"Employee Name:emp_name," &
   + "Department:dept_name,Employee Number:emp_num")
```

See also

f_sort_order
w_sort

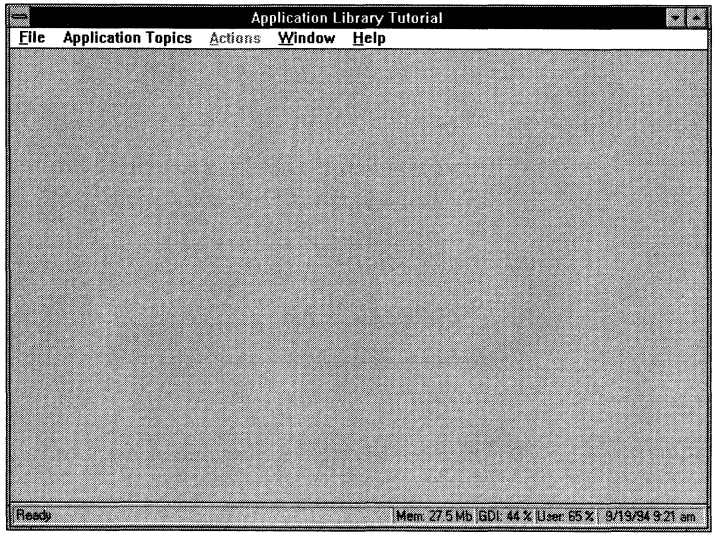
w_sys_frame

Description

Application framework window object that is the ancestor MDI frame window used for all MDI applications.

Use as an ancestor window
This window is designed to be used as the ancestor frame window for all MDI applications built using the Application Library application framework.

You can inherit from w_sys_frame for MDI frame windows or copy it. If inheritance is used, any changes made to w_sys_frame will affect all application frame windows inherited from it. Typically, changes are not necessary for w_sys_frame.



Type

MDI frame with MicroHelp

Library

SYS.PBL

Invocation

Open (*w_sys_frame_descendant*)

Parameter	Description
w_sys_frame_descendant	MDI frame window that is a descendant of w_sys_frame

Window events

Event	Description
Move (user event)	Indicates to the w_mdi_clock window that the parent window has been moved
Open	Registers the window as a global variable and opens the window w_mdi_clock
post_open (user event)	Calls the f_app_open function, which initializes instance variables and calls the f_login function

Event	Description
Resize	Indicates to the w_mdi_clock window that the parent window has been resized

Window functions

wf_allow_multiple Tells you whether the application allows multiple window instances, as specified in the f_app_open function.

- *Parameters.* None.
- *Return value.* Boolean. Returns TRUE if multiple instances are allowed and FALSE if they are not.

wf_application Returns the application object.

- *Parameters.* None.
- *Return value.* Application. Returns the application, as specified in the f_app_open function.

wf_ini_file_name Returns the name of the application INI file.

- *Parameters.* None.
- *Return value.* String. Returns the INI file name, as specified in the f_app_open function.

Control events

This window has no controls. But it does display the w_mdi_clock window at the bottom of the window.

For more this window, see the w_mdi_clock discussion on page 148

Instance variables

Variable	Data type	Access
ib_mult	Boolean	Private
is_ini_file	String	Private
ia_app	Application	Private
iw_frame	w_sys_frame	Public

Usage

Use the w_sys_frame window as the ancestor object for your MDI application's frame window. Use a descendant of the m_sys_frame menu as the menu for w_sys_frame.

Use with the m_sys_frame menu
Associate your descendent frame window with a descendant of the m_sys_frame menu and inherit from your frame menu to create menus for sheets opened in this frame. Keeping the m_sys_frame menu in your menu inheritance chain provides integration with user events, window functions, and user object functions defined in the application framework sheet windows.

Example

For an example of using this window as an ancestor object, see Lesson 2 of the tutorial application.

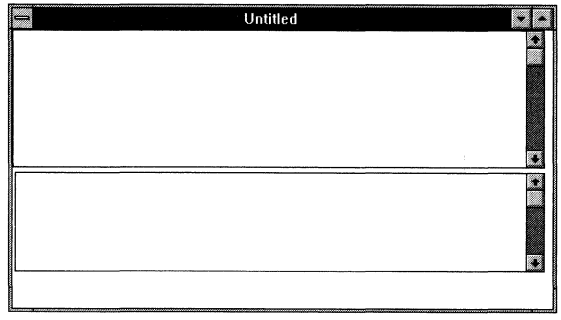
See also

m_sys_frame
w_mdi_clock
w_sys_mast_detl_dw
w_sys_multi_dw
w_sys_report
w_sys_shared_dw
w_sys_single_dw

w_sys_mast_detl_dw

Description

Application framework window object that provides the functionality needed for a many-to-many Master/Detail maintenance window. For a many-to-one Master/Detail window use the w_sys_shared_dw window.



Type

Main (but designed to be opened as a sheet in an MDI frame)

Library SYS.PBL

Invocation Open (*w_sys_mast_detl_dw_descendant*)

Parameter	Description
w_sys_mast_detl_dw_descendant	Window that is a descendant of w_sys_mast_detl_dw

Controls used

Control	Control type
dw_master	uo_dw standard user object (DataWindow control)
dw_detail	uo_dw standard user object (DataWindow control)

Control events for

Control	Event	Description
dw_master	GetFocus	Triggers the ItemFocusChanged event and makes sure the MicroHelp is updated.
dw_master	ItemFocusChanged	Sets the MicroHelp to the tag values assigned in the DataWindow.
dw_master	RetrieveEnd	Triggers the RowFocusChanged event to properly populate the detail DataWindow.
dw_master	RowFocusChanged	If there are no pending updates, this event changes rows. If updates have not been committed, it prompts the user to commit before moving the row indicator. You must add script to this event in your descendent window to retrieve on the detail DataWindow based upon the now current master row.
dw_detail	GetFocus	Triggers the ItemFocusChanged event and makes sure the MicroHelp is updated.
dw_detail	ItemFocusChanged	Sets the MicroHelp to the tag values assigned in the DataWindow.
dw_detail	RetrieveEnd	Triggers the RowFocusChanged event to properly show the current row in the detail DataWindow.

Control	Event	Description
dw_detail	RowFocusChanged	Highlights the current row.

Window events

Event	Description
Close	If duplicate instances of the same information cannot be opened, this event removes the sheet from the list of open windows for this type of sheet.
CloseQuery	Checks the DataWindows to see if any rows have been modified and not saved. If so, it calls the f_exit_status function, which opens the w_exit_status window allowing the user to save the modifications and exit the window, exit without saving, or cancel the request.
Open	Sets the transaction object for the dw_master and dw_detail DataWindow controls. It also registers the MDI frame assigned to the sheet. This registration assigns the MDI frame to an instance variable, allowing you to reference it in scripts without hardcoding the actual window name.
Resize	Resizes the DataWindow controls dw_master (width only) and dw_detail (width and height).
ue_filenew (user event)	Checks to see if the current information displayed was modified and prompts the user to save the information. Then it resets the dw_detail control and inserts a new row into the dw_master control. This event is triggered from the File➤New menu item or the corresponding toolbar button.
ue_fileopen (user event)	Used to open a search window and display information based on entered criteria. This event is triggered from the File➤Open menu item or from the associated toolbar button. The user is prompted to save any changes that have been made before the new search criteria are selected. This user event triggers the ue_select_window user event, which you can use to perform processing that selects new information (usually by opening a descendant of the w_select window). This information is then stored in instance variables referenced in the ue_retrieve_data event, which performs the actual retrieve. You must code the ue_select_window and ue_retrieve_data events in the descendent window.

Event	Description
ue_filedelete (user event)	Prompts for confirmation on the delete. If yes, it triggers the ue_validatedelete user event. If the row can be deleted, it first deletes detail rows and then deletes master rows. This event does an automatic update to the database and is triggered from the File➤Delete menu item or the corresponding toolbar button.
ue_filesave (user event)	Makes sure that the information entered in the DataWindow controls has been accepted and then saves it. This event is triggered from the File➤Save menu item or from the corresponding toolbar button.
ue_filesaveas (user event)	Calls the PowerBuilder SaveAs function, which allows the information displayed in the DataWindows to be saved as a file of a user-specified type. This event is triggered from the File➤SaveAs menu item or the corresponding toolbar button.
ue_fileprint (user event)	Calls the PowerBuilder Print function, which prints both DataWindows. This event can be overridden in the descendant to call other print functions. This event is triggered from the File➤Print menu item or the corresponding toolbar button.
ue_validate (user event)	Not used in the ancestor window. Descendent windows can trigger this event to perform cross-reference checking between multiple columns before the information is saved.
ue_validatedelete (user event)	Called from the ue_filedelete user event. The event initializes the boolean variable ib_data_ok = TRUE. The descendent event scripts should include referential integrity checking as needed. If the delete should not be performed for referential integrity reasons, the descendent script should set ib_data_ok to FALSE.
ue_select_window (user event)	Not used in the ancestor window. Call this event from the ue_fileopen and ue_open_as_dependent events in the descendent window. Use it to set instance variables that you then use in the ue_retrieve_data event to retrieve data into the DataWindow controls. For examples, see the Pubs sample application.

Event	Description
ue_open_as_dependent (user event)	Similar to the ue_fileopen event in that is also triggers the ue_select_window and ue_retrieve_data events. You use the ue_select_window to populate the instance variable(s) used in the ue_retrieve_data event without prompting the user for the information. Trigger this event in the window Open event script when being opened from another window if multiple instances are not allowed.
ue_fileprint_preview (user event)	Opens the w_printzoom window, which allows the user to specify the degree of zoom and other options for viewing the DataWindow before printing it. Both DataWindows are affected by the degree of zoom.
ue_retrieve_data (user event)	Not used in the ancestor window. Use this event to retrieve data by coding a dw_sheet.Retrieve function in the descendent window.
ue_detail_delete (user event)	Prompts for confirmation on the delete of a detail row. If yes, it triggers the user defined event ue_validatedelete_detail. If the row can be deleted, the delete is performed. This event is triggered from the Action➤Delete Detail menu item or from the corresponding toolbar button.
ue_detail_new (user event)	Inserts a new row into the detail DataWindow. Make sure that code is placed in the descendant to populate any related columns from the master DataWindow. For an example, see the w_author_master_detail_sheet in the Pubs sample application.
ue_sort_master (user event)	Opens the window w_sort, which allows the user to specify a sort order for the master DataWindow.
ue_sort_detail (user event)	Opens the window w_sort, which allows the user to specify a sort order for the detail DataWindow.

Window functions

wf_modified Returns the status of the DataWindow controls on the window.

- *Parameters.* None.
- *Return value.* Integer. Returns *-1* if either DataWindow fails the AcceptText function (the data doesn't pass validation), *0* if rows have been deleted or updated, and *1* if all changes have been saved to the database.

wf_ok_to_continue Checks to see if data was modified, and if so uses the f_exit_status function to allow the user to choose the action to be taken.

- *Parameters.* String specifying whether data should be saved because the user is exiting or simply changing DataWindow rows:
  - **M** specifies the user is moving between rows
  - **S** specifies the user is exiting the window

  This parameter is used by the f_exit_status function, which can be called by this event.
- *Return value.* Boolean. Returns TRUE if processing should continue and FALSE if the current window should be maintained.

wf_set_sheettitle If the application does not allow windows with duplicate information to be opened at the same time, this window function determines if the instance of the window being opened has already been opened by checking the window title against a shared string array variable. If the title already exists, the function returns FALSE; if it does not exist, the title is added to the list of existing titles.

- *Parameters.* String containing the information being added to the title of the window.
- *Return Value.* Boolean. Returns TRUE if no window instance is already open and FALSE if a window instance is already open.

wf_update_dw Updates the database and does all error checking. If the master DataWindow is performing a delete, then the update for the detail DataWindow is performed first and then the master DataWindow is updated. This is done to make sure that any referential integrity checking done by the database will not cause the transaction to roll back because the master record was deleted with the detail record still existing.

If it is not for a delete, then the update for the master is performed first to ensure that the primary key exists in the database before any records are inserted from the detail DataWindow that uses that primary key.

If either of the DataWindows fails to successfully perform its update, then all updates are rolled back and the DataWindows remain in the state that they were in (they are not aware that an update has been attempted) before the update process started.

- *Parameters.* None.

- *Return Value.* Boolean. Returns TRUE if the update succeeded and FALSE if it failed.

Instance variables

Variable	Data type	Access
ib_data_ok	Boolean	Protected
ib_inserting	Boolean	Protected
ib_insert_on_open	Boolean	Protected
ib_insert_on_zero_rows	Boolean	Protected
il_master_cur_row	Long	Protected
il_detail_cur_row	Long	Protected
is_window_title	String	Protected
is_window_title_new	String	Protected
is_window_title_data	String	Protected
allow_duplicate_processes	Boolean	Protected
iw_frame	w_sys_frame	Protected
str_parms	istr_parms	Protected

Usage

Use descendants of this window for all sheets that display two DataWindow controls with a many-to-many master/detail relationship. Its frame window should be a descendant of w_sys_frame. In addition to your application-specific processing, you must add script to retrieve rows (in the ue_retrieve_data user event) and to synchronize the DataWindow controls (in the dw_master RowFocusChanged event).

Associate this sheet window with a descendant of your frame menu. The frame menu should in turn be a descendant of m_sys_frame. Keeping the m_sys_frame menu in your inheritance chain provides integration with user events, window functions, and user object functions defined in this window.

Referential integrity
This window assumes that the DBMS handles referential integrity automatically. That is, it does not lock the master row when updating a related detail row.

Example

For an example of using w_sys_mast_detl_dw as an ancestor window, see the w_author_master_detail_sheet window in the Pubs sample application.

See also

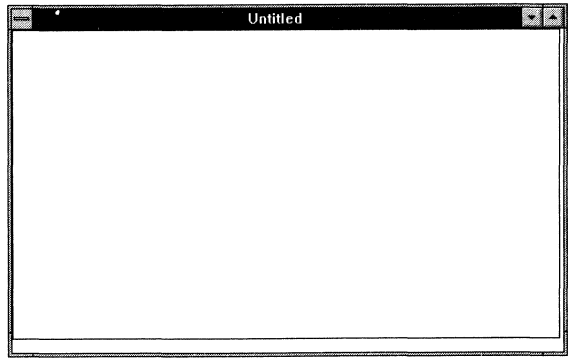
f_db_error
f_exit_status
f_get_token
w_sys_frame
w_sys_multi_dw
w_sys_report
w_sys_shared_dw
w_sys_single_dw

w_sys_multi_dw

Description

Application framework window object that provides the functionality needed for multirow DataWindows. This window is the ancestor for all windows that contain multirow DataWindows.

This window is inherited from w_sys_single_dw, described on page 193. Additionally, it is the ancestor for w_sys_shared_dw, described on page 191.



Type

Main (but designed to be opened as a sheet in an MDI frame)

Library

SYS.PBL

Invocation

Open (*w_sys_multi_dw_descendant*)

Parameter	Description
w_sys_multi_dw_descendant	Window that is a descendant of w_sys_multi_dw

Controls used

Control	Control type
dw_sheet	uo_dw standard user object (DataWindow control)

Control events

Control	Event	Description
dw_sheet	RowFocusChanged	Highlights the current row

Window events

Event	Description
ue_filenew (user event)	Overrides the ancestor script. This event inserts a row after the current row of the DataWindow unless no rows exist. This event is triggered from the File➤New menu item or the corresponding toolbar button.
ue_filedelete (user event)	Overrides the ancestor script. This event deletes one row at a time from the multirow DataWindow. This event is triggered from the File➤Delete menu item or the corresponding toolbar button.
ue_deleteall (user event)	Deletes all rows from the multirow DataWindow.

Inherited events are not documented in this section. For information on inherited events, see "w_sys_single_dw," on page 193.

Window functions

wf_deleterow This function is called from the ue_deleteall user event. As each row of the DataWindow is about to be deleted, the row number is passed to this function. If referential integrity checks have passed, this function calls the DeleteRow function for the row. To add referential integrity checks to your application, define them using the uf_add_validation user object function, explained in Chapter 9, "User Objects."

- *Parameters.* The current row number.
- *Return value.* Boolean. TRUE indicates that the delete was successful; FALSE indicates that it failed.

Instance variables

Variable	Data type	Access
il_current_row	Long	Protected
il_new_row	Long	Protected
il_num_rows	Long	Protected

Usage

Use descendants of this window for all sheets that display a multirow DataWindow control. Its frame window should be a descendant of w_sys_frame. In addition to your application-specific processing, you must add script to retrieve rows (in the ue_retrieve_data user event).

Associate this sheet window with a descendant of your frame menu. The frame menu should in turn be a descendant of m_sys_frame. Keeping the m_sys_frame menu in your inheritance chain provides integration with user events, window functions, and user object functions defined in this window.

Example

For an example of using w_sys_multi_dw as an ancestor window, see the w_titleauthor_sheet window in the Pubs sample application.

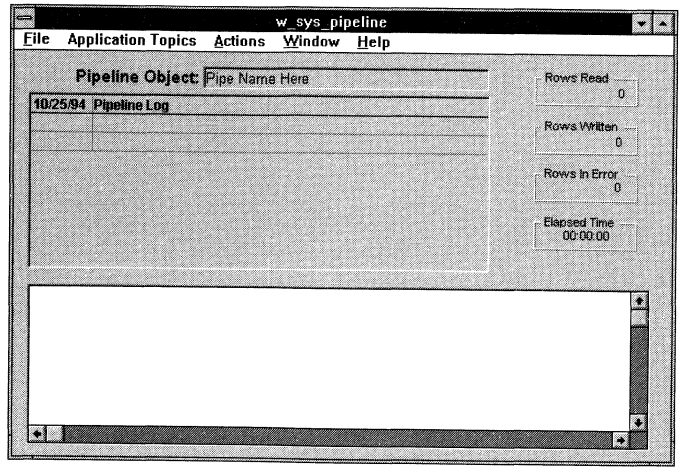
See also

w_sys_frame
w_sys_mast_detl_dw
w_sys_report
w_sys_shared_dw
w_sys_single_dw

w_sys_pipeline

Description

Application framework window object that provides the functionality needed to execute a data pipeline. This window creates an instance of the u_pipeline_kit Class user object (nonvisual user object) and provides events, user events, and windows functions that allow you to move data between data sources using a pipeline. This window is the ancestor for all windows that provide pipeline functionality.



Type

Main (but designed to be opened as a sheet in an MDI frame)

Library

SYS.PBL

Invocation

Open (*w_sys_pipeline_descendant*)

Parameter	Description
w_sys_pipeline_descendant	Window that is a descendant of w_sys_pipeline

Controls used

Control	Control type
dw_msg	DataWindow that displays processing messages
dw_pipe_errors	DataWindow used by the pipeline to display errors
st_elapsed_time	StaticText to display elapsed time
st_errors	StaticText to display the number of errors that can occur before the pipeline stops processing
st_read	StaticText to display the number of rows read
st_written	StaticText to display the number of rows written

Window events

Event	Description
CloseQuery	Checks the ib_executing instance variable to determine if the pipeline is currently executing. If the pipeline is executing, this event sets Message.ReturnValue to 1, stopping window Close processing.
Open	The descendent window's Open event must: ♦ Create a transaction object for the source database ♦ Create a transaction object for the destination database ♦ Create the pipeline object and connect to the databases by calling the wf_initialize_pipeline function
ue_cancel_pipe (user event)	Cancels pipeline execution. This event is triggered from the Actions➤Cancel menu item.
ue_execute_pipe (user event)	Begins pipeline execution. This event is triggered from the Actions➤Execute menu item.
ue_print_errors (user event)	Prints the pipeline errors DataWindow. This event is triggered from the File➤Print Errors menu item.
ue_print_log (user event)	Prints the dw_msg DataWindow. This event is triggered from the File➤Print Log menu item.
ue_repair_pipe (user event)	Resubmits corrected rows to the destination database. This event is triggered from the Actions➤Repair Pipeline menu item.
ue_reset_log (user event)	Clears the rows in the dw_msg DataWindow. This event is triggered from the File➤Reset Log menu item.
ue_set_pipe_commit (user event)	Sets the pipeline COMMIT frequency to the value specified in Message.LongParm. This event is triggered from choices under the Actions➤Pipeline Commits menu item.
ue_set_pipe_ext_attr_copy (user event)	Controls whether the pipeline copies extended attributes by setting the attribute to the value specified in Message.StringParm. This event is triggered from the Actions➤Pipeline Copy Extended Attributes menu item.

Event	Description
ue_set_pipe_maxerrors (user event)	Sets the value of the attribute indicating the number of errors the pipeline will allow before canceling processing. This event is triggered from choices under the Actions➤Pipeline Maximum Errors menu item.
ue_set_pipe_type (user event)	Sets the pipeline type (create, append, replace, and so on). This event is triggered from choices under the Action➤Pipeline Type menu item.

Window functions

wf_add_msg Adds a row to the dw_msg DataWindow.

- *Parameters.* String specifying the message text to add to the dw_msg DataWindow.
- *Return value.* None.

wf_get_errcount Returns the number of errors.

- *Parameters.* None.
- *Return value.* Long indicating the number of rows in the dw_pipe_errors DataWindow.

wf_get_pipe_ext_attr_copy Returns a boolean indicating whether extended attributes will be copied.

- *Parameters.* None.
- *Return value.* Boolean. TRUE if extended attributes will be copied and FALSE if they will not.

wf_get_pipe_commit Returns the current COMMIT frequency.

- *Parameters.* None.
- *Return value.* Long indicating the current COMMIT frequency.

wf_get_pipe_maxerrors Returns the value of the attribute indicating the number of errors the pipeline will allow before canceling execution.

- *Parameters.* None.
- *Return value.* Long indicating the maximum errors the pipeline will allow before canceling execution.

wf_get_pipe_type Returns the pipeline type.

- *Parameters.* None.
- *Return value.* String indicating the pipeline type.
  - Create
  - Replace
  - Append
  - Refresh
  - Update

wf_initialize_pipeline Initializes the u_pipeline_kit user object and connects to the source and destination databases. For sample code that you place in the Open event to prepare the call to wf_initialize_pipeline, see the comments in the w_sys_pipeline Open event.

- *Parameters*:
  - String specifying the name of the pipeline object.
  - Transaction object for the source database (passed by reference). You must create this transaction object before calling the wf_initialize_pipeline function.
  - Transaction object for the destination database (passed by reference). You must create this transaction object before calling the wf_initialize_pipeline function.
- *Return value.* Boolean. Returns TRUE if the database connection succeeded and FALSE if it did not.

wf_set_menu_attributes Sets menu item states to reflect the current state of the pipeline object.

- *Parameters.* None.
- *Return value.* None.

Instance variables

Variable	Data type	Access
ib_attempt_connect	Boolean	Public
ib_executing	Boolean	Public
i_dest	Transaction	Public
i_pipe	U_pipeline_kit	Public

Variable	Data type	Access
i_src	Transaction	Public

Usage

Use descendants of this window for all sheets that implement data pipeline functionality. To use this functionality, you must also have defined:

- A data source for the source database
- A data source for the destination database
- A pipeline object (using the Data Pipeline painter)

> **Pipeline object versus pipeline user object**
> A pipeline *object* includes data selection criteria, source database information, and destination database information. You create pipeline objects using the Data Pipeline painter. A pipeline *user object* is a Class object descended from the Pipeline system object. The application framework includes the u_pipeline_kit pipeline user object. At runtime, you create an instance of the u_pipeline_kit user object and associate it with a pipeline object using the wf_initialize_pipeline window function.

Example

To use the w_sys_pipeline, window:

1. In the Window painter, create a descendant of w_sys_pipeline.
2. Use the PowerScript painter to access the descendent window's Open event. In this event, you include PowerScript statements that:
   - Create a transaction object for the source database.
   - Create a transaction object for the destination database.
   - Call the wf_initialize_pipeline window function, passing the name of the pipeline object, the source transaction object, and the destination transaction object. This function creates an instance of the pipeline user object and connects to the databases.
3. Save the window.
4. In the Menu painter, create a descendant of m_sys_frame.
5. Enable pipeline functionality in the File and Actions menus.
6. Save the menu.
7. Use the Window painter to associate the new menu with your w_sys_pipeline descendant.

8 Associate your new window with a frame menu by adding a menu item to the frame menu that opens it as an MDI sheet.

9 Test the pipeline window.

These instructions assume that a pipeline object, source data source, and destination data source already exist.

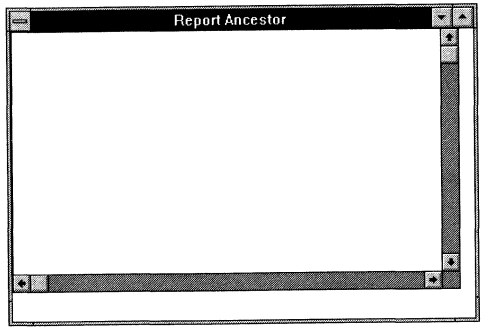
See also

m_sys_frame
w_sys_frame
u_pipeline_kit

w_sys_report

Description

Application framework window object that provides the functionality needed for generating reports where the report is a single DataWindow. It allows you to automatically implement query mode (set the corresponding query mode menu items to visible and enabled) and also limit which columns can be used with query mode. It also includes print preview and zoom functionality. It has no data manipulation or update functionality and assumes that all DataWindow columns have a tab order of zero.



Type

Main (but designed to be opened as a sheet in an MDI frame)

Library

SYS.PBL

Invocation

Open (*w_sys_report_descendant*)

Parameter	Description
w_sys_report_descendant	Window that is a descendant of w_sys_report

Controls used

Control	Control type
dw_sheet	DataWindow

Window events

Event	Description
Close	If duplicate instances of the same information cannot be opened, this event removes the sheet from the list of open windows for this type of sheet.
Open	Sets the transaction object for the DataWindow control dw_sheet and also registers the MDI frame and the menu assigned to the sheet. This registration assigns the menu and MDI frame to instance variables, allowing you to reference them in scripts without hardcoding actual menu and window names.
Resize	Resizes the dw_sheet DataWindow control to the size of the window.
ue_filesaveas (user event)	Calls the PowerBuilder SaveAs function, which allows the information displayed in the DataWindow to be saved as a file of a user-specified type. This event is triggered from the File➤SavcAs menu item or the corresponding toolbar button.
ue_fileprint (user event)	Calls the PowerBuilder Print function, which prints the DataWindow. This event can be overridden in the descendant to call other print functions. This event is triggered from the File➤Print menu item or the corresponding toolbar button.
ue_retrieve_data (user event)	Not used in the ancestor window. Use this event to retrieve data by coding a dw_sheet.Retrieve function in the descendent window.
ue_select_window (user event)	Not used in the ancestor window. Call this event from the ue_fileopen and ue_open_as_dependent events. Use it to set instance variables that you then use in the ue_retrieve_data event to retrieve data into the DataWindow. For examples, see the Pubs sample application.

Event	Description
ue_open_as_dependent (user event)	Triggers the ue_select_window and ue_retrieve_data events. You use the ue_select_window event to populate the instance variable(s) used in the ue_retrieve_data event without prompting the user for the information. Trigger this event in the window Open event script when being opened from another window if multiple instances are not allowed.
ue_fileprint_preview (user event)	Opens the w_printzoom window, which allows the user to specify the degree of zoom and other options for viewing the DataWindow before printing it.
ue_zoom_smaller (user event)	Decrements the zoom factor (stored in the ii_current_zoom instance variable) by the value stored in the ii_zoom_increment instance variable and then redisplays the dw_sheet DataWindow using that zoom factor.
ue_zoom_bigger (user event)	Increments the zoom factor (stored in the ii_current_zoom instance variable) by the value stored in the ii_zoom_increment instance variable and then redisplays the dw_sheet DataWindow using that zoom factor.
ue_toggle_query_mode (user event)	Examines the dw_sheet DataWindow; if the DataWindow is not in query mode it sets the border and tab sequence of the columns that have been marked as allowable for query mode so that the user can enter data to be used for the query; puts the DataWindow into query mode; and resets the menu and toolbar to reflect query mode. If the DataWindow is already in query mode, the event resets the columns used for query mode input, takes the DataWindow out of query mode, and triggers the ue_retrieve_data event. For information on specifying columns for query mode, see the wf_add_query_mode_column function, under Window functions, next.
ue_reset_query_criteria (user event)	Resets the DataWindow so that any data it contains is eliminated. It also resets any information that may have been entered to be used as query criteria.
ue_sort	Opens the w_sort window, allowing the user to specify sort criteria for the DataWindow.

Window functions

wf_set_sheettitle If the application does not allow windows with duplicate information to be opened at the same time, this window function determines if an instance of the window being opened is already open. It does this by checking the window title against a shared string array variable. If the title already exists, the function returns FALSE; if it does not exist, the title is added into the list of existing titles.

- *Parameters.* String containing the information being added to the title of the window.
- *Return value.* Boolean. Returns TRUE if no window instance is already open and FALSE if a window instance is already open.

wf_query_status Returns the status of the DataWindow's query mode attributes.

- *Parameters.* None.
- *Return Value.* Boolean. Returns TRUE if the DataWindow is in query mode and FALSE if it is not.

For more information on query mode, see *Building Applications* in the PowerBuilder documentation set.

wf_set_query_state Turns query mode on or off, as specified by the input parameter. It also sets the menu state to reflect the current mode.

- *Parameters.* Boolean specifying the requested query mode state. TRUE turns query mode on and FALSE turns it off.
- *Return value.* None.

wf_add_query_mode_column Adds to the list of columns that are allowed to be used while in query mode.

- *Parameters.* String specifying the name of the column to be added to the list.
- *Return value.* None.

Instance variables

Variable	Default	Data type	Access
is_window_title	None	String	Protected
is_new_window_title	None	String	Protected
is_window_title_data	None	String	Protected

Variable	Default	Data type	Access
allow_duplicate_processes	None	Boolean	Protected
ib_data_ok	None	Boolean	Protected
istr_parms	None	str_parms	Protected
im_menu_id	None	m_sys_frame	Protected
iw_frame	None	w_sys_frame	Protected
ii_zoom_increment	10	Integer	Protected
ii_current_zoom	None	Integer	Protected
is_query_mode_cols[]	None	String	Protected

Usage

Use descendants of this window for all report sheets. Its frame window should be a descendant of w_sys_frame. In addition to your application specific processing, you must add script to retrieve rows (in the ue_retrieve_data user event).

Associate this sheet window with a descendant of your frame menu. The frame menu should in turn be a descendant of m_sys_frame. Keeping the m_sys_frame menu in your inheritance chain provides integration with user events, window functions, and user object functions defined in this window.

Example

For an example of using this window as an ancestor object, see any of the online reports in the Time Management sample application.

See also

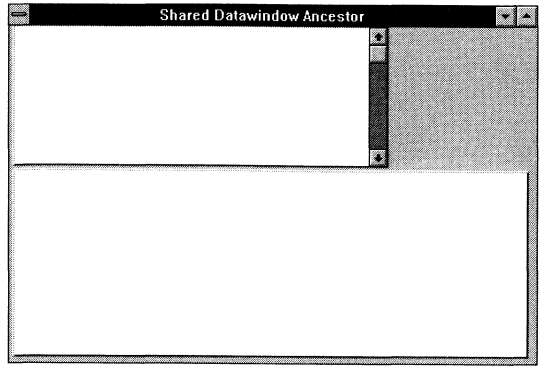
w_printzoom
w_sort
w_sys_frame
w_sys_mast_detl_dw
w_sys_multi_dw
w_sys_shared_dw
w_sys_single_dw

w_sys_shared_dw

Description

Application framework window object that provides the functionality needed for a many-to-one Master/Detail maintenance window. For many-to-many Master/Detail windows, use the w_sys_mast_detl_dw window.

This window is a descendant of w_sys_multi_dw, described on page 178.



Type

Main (but designed to be opened as a sheet in an MDI frame)

Library

SYS.PBL

Invocation

Open (*w_sys_shared_dw_descendant*)

Parameter	Description
w_sys_shared_dw_descendant	Window that is a descendant of w_sys_shared_dw

Controls used

Control	Control type
dw_sheet	uo_dw standard user object (DataWindow control)
dw_detail	DataWindow control

Control events

Control	Event	Description
dw_sheet	RowFocusChanged	Causes the dw_detail DataWindow control to scroll to the current row of the dw_sheet DataWindow control

Control	Event	Description
dw_detail	RowFocusChanged	Causes the dw_sheet DataWindow control to scroll to the current row of the dw_detail DataWindow control

Window events

Event	Description
open	Extends the ancestor script by sharing the two DataWindows and then posts the ue_retrieve_data event.
Resize	Overrides the ancestor script. No action is taken when the window is resized.
ue_filenew (user event)	Overrides the ancestor script. It inserts a row after the current row of the DataWindow unless no rows exist. This event is triggered from the File➤New menu item or the corresponding toolbar button.

Inherited events are not documented in this section. For information on inherited events, see "w_sys_single_dw," on page 193 and "w_sys_multi_dw," on page 178.

Window functions

wf_modified Overrides the ancestor and includes functionality that examines the dw_detail DataWindow when determining if unsaved changes exist.

- *Parameters.* None.
- *Return value.* Integer. Returns *-1* if the DataWindow fails the AcceptText function (the data doesn't pass validation), *0* if rows have been deleted or updated, and *1* if all changes have been saved to the database.

Usage

Use descendants of this window for all sheets that display two DataWindow controls with a many-to-one Master/Detail relationship. Its frame window should be a descendant of w_sys_frame. In addition to your application-specific processing, you must add script to retrieve rows for the dw_sheet DataWindow control (in the ue_retrieve_data user event).

Associate this sheet window with a descendant of your frame menu. The frame menu should in turn be a descendant of m_sys_frame. Keeping the m_sys_frame menu in your inheritance chain provides integration with user events, window functions, and user object functions defined in this window.

Example

For an example of using descendants of this window, see the tutorial in Part Two.

See also

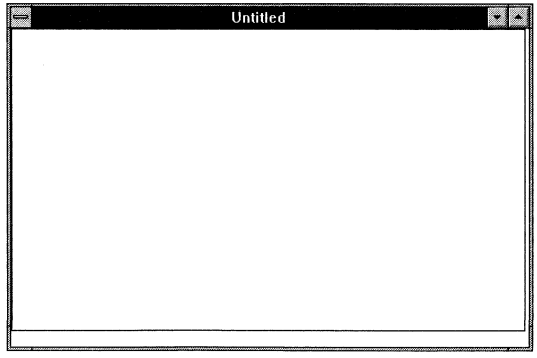
w_sys_frame
w_sys_mast_detl_dw
w_sys_multi_dw
w_sys_report
w_sys_single_dw

w_sys_single_dw

Description

Application framework window object that provides the functionality needed for a single-row maintenance window. In the sample applications, this window is used as the ancestor for all windows with a single-row DataWindow.

This window is the ancestor for w_sys_multi_dw, described on page 178.



Type

Main (but designed to be opened as a sheet in an MDI frame)

Library

SYS.PBL

Invocation

Open (*w_sys_single_dw_descendant*)

Parameter	Description
w_sys_single_dw_descendant	Window that is a descendant of w_sys_single_dw

Controls used

Control	Control type
dw_sheet	uo_dw standard user object (DataWindow control)

Control events

Control	Event	Description
dw_sheet	ItemFocusChanged	Sets the MicroHelp to the tag values assigned in the DataWindow.
dw_sheet	RetrieveEnd	Resets the toolbar buttons and menu items based on the status of the DataWindow.
dw_sheet	ue_tab_out (user event)	Not used in the ancestor window. Descendent windows should set the column in this event to the first column in the DataWindow. This event is triggered by the pbm_dwntabout event for the DataWindow.
dw_sheet	on_delete (user event)	Triggered by the SQLPreview event. It allows you to perform application-specific processing just before deleting a row from the database. To reject the request, issue a SetActionCode(1) function.
dw_sheet	on_insert (user event)	Triggered by the SQLPreview event. It allows you to perform application-specific processing just before inserting a row into the database. To reject the request, issue a SetActionCode(1) function.
dw_sheet	on_update (user event)	Triggered by the SQLPreview event. It allows you to perform application-specific processing just before updating a row in the database. To reject the request, issue a SetActionCode(1) function.

Window events

Event	Description
Close	If duplicate instances of the same information cannot be opened, this event removes the sheet from the list of open windows for this type of sheet.
CloseQuery	Checks the DataWindow to see if any rows have been modified and not saved. If any have, the f_exit_status function is called, which opens the w_exit_status window, allowing the user to save the modifications and exit the window, exit without saving, or cancel the request.
Open	Sets the transaction object for the DataWindow control dw_sheet and also registers the MDI frame and the menu assigned to the sheet. This registration assigns the menu and MDI frame to instance variables, allowing you to reference them in scripts without hardcoding actual menu and window names.
Resize	Resizes the DataWindow control dw_sheet to the size of the window.
ue_filenew (user event)	Checks to see if the current information displayed was modified and prompts the user to save the information. Then it resets DataWindow and inserts a new row. This event is triggered from the File➤New menu item or from the corresponding toolbar button.
ue_fileopen (user event)	Used to open a search window and display information based on entered criteria. The user is prompted to save any changes that have been made before the new record is selected. This event is triggered from the File➤Open menu item or the corresponding toolbar button. This user event triggers the ue_select_window user event, which is expected to perform processing that selects new information (usually by opening a descendant of the w_select window). This information is then stored in instance variables that are referenced in the ue_retrieve_data event, which performs the actual retrieve. You must code the ue_select_window and ue_retrieve_data events in the descendent window.

Event	Description
ue_filedelete (user event)	Prompts for confirmation on the delete. If yes, it triggers the ue_validatedelete user event. If the row can be deleted, the delete is performed. This event does an automatic update to the database and is triggered from the File➤Delete menu item or the corresponding toolbar button.
ue_filesave (user event)	Ensures that the information entered in the DataWindow control has been accepted and performs the update. This event is triggered from the File➤Save menu item or the corresponding toolbar button.
ue_filesaveas (user event)	Calls the PowerBuilder SaveAs function, which allows the information displayed in the DataWindow to be saved as a file of a user-specified type. This event is triggered from the File➤SaveAs menu item or the corresponding toolbar button.
ue_fileprint (user event)	Calls the PowerBuilder Print function, which prints the DataWindow. This event can be overridden in the descendant to call other print functions. This event is triggered from the File➤Print menu item or the corresponding toolbar button.
ue_retrieve_data (user event)	Not used in the ancestor window. Use this event to retrieve data by coding a dw_sheet.Retrieve function in the descendent window.
ue_validate (user event)	Not used in the ancestor window. Descendent windows can trigger this event to perform cross-reference checking between multiple columns before the information is saved.
ue_validatedelete (user event)	Called from the ue_filedelete user event. The event initializes the boolean variable ib_data_ok to TRUE. The descendent event scripts should include referential integrity checking as needed. If the delete should not be performed for referential integrity reasons, the descendent script should set the ib_data_ok variable to FALSE.
ue_select_window (user event)	Not used in the ancestor window. Call this event from the ue_fileopen and ue_open_as_dependent events in the descendant window. Use it to set instance variables that you then use in the ue_retrieve_data event to retrieve data into the DataWindow. For examples, see the Pubs sample application.

Event	Description
ue_open_as_dependent (user event)	Similar to the ue_fileopen event in that it also triggers the ue_select_window and ue_retrieve_data events. You use the ue_select_window to populate the instance variable(s) used in the ue_retrieve_data event without prompting the user for the information. Trigger this event in the window Open event script when being opened from another window if multiple instances are not allowed.
ue_fileprint_preview (user event)	Opens the w_printzoom window, which allows the user to specify the degree of zoom and other options for viewing the DataWindow before printing it.

Window functions

wf_set_sheettitle If the application does not allow windows with duplicate information to be opened at the same time, this window function determines if an instance of the window being opened is already open by checking the window title against a shared string array variable. If the title does not exist, it adds the title to the list of existing titles.

- *Parameters.* String containing the information being added to the title of the window.
- *Return value.* Boolean. Returns TRUE if no window instance is already open and FALSE if a window instance is already open.

wf_update_dw Performs the update to the DataWindow control and does all error checking.

- *Parameters.* None.
- *Return value.* Boolean. Returns TRUE if the update succeeded and FALSE if it failed.

wf_modified Returns the status of the window's DataWindow control.

- *Parameters.* None.
- *Return value.* Integer. Returns *-1* if the DataWindow fails the AcceptText function (the data doesn't pass validation), *0* if rows have been deleted or updated, and *1* if all changes have been saved to the database.

Instance variables

Variable	Data type	Access
allow_duplicate_processes	Boolean	Protected
ib_data_ok	Boolean	Protected
ib_insert_on_open	Boolean	Protected
im_menu_id	m_sys_frame	Protected
is_new_window_title	String	Protected
is_window_title	String	Protected
is_window_title_data	String	Protected
istr_parms	str_parms	Protected
iw_frame	w_sys_frame	Protected

Usage

Use descendants of this window for all sheets that display a single-row DataWindow control. Its frame window should be a descendant of w_sys_frame. In addition to your application-specific processing, you must add script to retrieve the row (in the ue_retrieve_data user event).

Associate this sheet window with a descendant of your frame menu. The frame menu should in turn be a descendant of m_sys_frame. Keeping the m_sys_frame menu in your inheritance chain provides integration with user events, window functions, and user object functions defined in this window.

Example

For an example of using this window as an ancestor object, see the w_author_sheet window in the Pubs sample application.

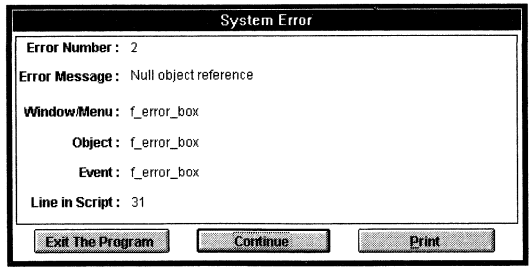
See also

f_db_error
f_exit_status
uo_dw
w_sys_frame
w_sys_mast_detl_dw
w_sys_multi_dw
w_sys_report
w_sys_shared_dw

w_system_error

Description

Displays system errors. It allows the user to continue running the application, exit the application, or print the error message. You should call it from the SystemError event in the application object.



Type

Response

Library

UTLWIN.PBL

Invocation

Open (**w_system_error**)

Controls used

Control	Control type
dw_error	DataWindow
cb_continue	CommandButton
cb_exit	CommandButton
cb_print	CommandButton

Control events

Control	Event	Description
cb_continue	Clicked	Closes the w_system_error window, remaining in the application
cb_exit	Clicked	Closes the w_system_error window, ending the application
cb_print	Clicked	Prints the error information displayed on the window

Window events

Event	Description
Open	Sets the error information from the error object into dw_error

Usage

Use this window in the application object's SystemError event to display system errors.

Example

This example opens window w_system_error from the SystemError event in the application object.

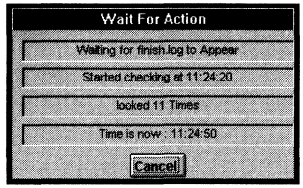
```
Open (w_system_error)
```

w_wait_for

Description

Checks every three seconds to determine if a file has appeared or disappeared.

The f_wait_for function opens this object.



Type

Response

Library

UTLWIN.PBL

Invocation

f_wait_for (*filename, appeardisappear*)

Parameter	Description
filename	String containing the name of the file

Parameter	Description
appeardisappear	Boolean variable indicating whether to wait for file to appear or to disappear: ♦ TRUE — Wait for the named file to appear ♦ FALSE — Wait for the named file to disappear

Controls used

Control	Control type
cb_cancel	CommandButton
st_curr_time	StaticText
st_filename	StaticText
st_start_time	StaticText
st_times_looked	StaticText

Control events

Control	Event	Description
cb_cancel	Clicked	Closes the window and returns FALSE to the calling function

Window events

Event	Description
Open	Initializes window text and triggers the Timer event.
Timer	Looks every three seconds for the filename passed to determine whether it has appeared or disappeared. Once the file exists or is deleted, the window will be closed, returning TRUE back to the calling function.

Instance variables

Variable	Data type	Access
ib_appear	Boolean	Public
is_filename	String	Public
ii_times_looked	Integer	Public

Usage

Use this window when your application depends on the presence or absence of a particular file.

Example

This example runs a process named work.pif, which writes a file named done.xxx when it is completed. The script will resume running when this file appears.

```
boolean status

// Make sure that the semaphore file does not exist.
FileDelete('done.xxx')
Run("work.pif")
// Wait for the file 'done.xxx' to appear.
status = f_wait_for('done.xxx',TRUE)
IF NOT status THEN
   // Error on process or user canceled.
END IF
```

See also

f_wait_for

CHAPTER 5

DataWindow Objects

About this chapter

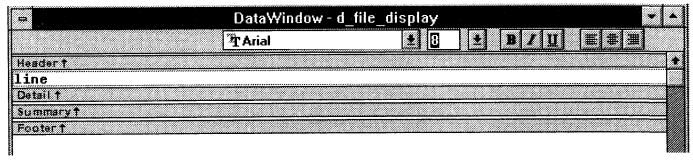
This chapter describes the DataWindow objects contained in the Application Library.

For information on the uo_dw DataWindow user object, which is a DataWindow control used in application framework windows to perform database access, refer to Chapter 9, "User Objects."

d_file_display

Description

Displays the contents of a previously stored text file in the w_file_display window.



Library

UTLWIN.PBL

Usage

This DataWindow object is used by the function f_display_file.

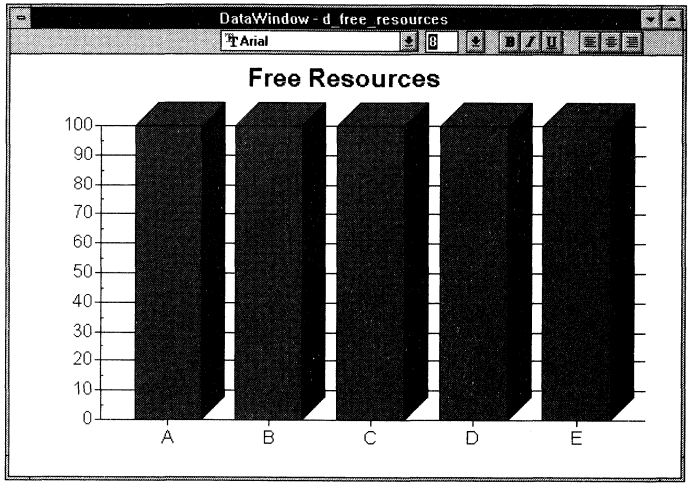
See also

f_display_file
w_display_file

d_free_resources

Description

Displays the free resources graph.



Library

UTLWIN.PBL

Usage

This DataWindow object is used by the window w_get_free_resources_graph.

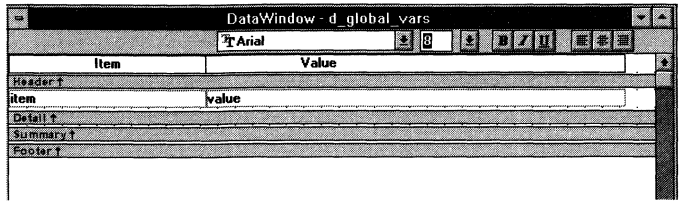
See also

w_get_free_resources
w_get_free_resources_graph

d_global_vars

Description

Maintains global variables.



Library

UTLFUNC.PBL

Usage

This DataWindow object is used in the window w_hold_parms.

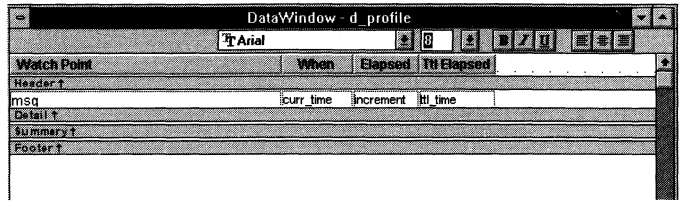
See also

w_hold_parms

d_profile

Description

Displays profile information.



Library

UTLWIN.PBL

Usage

This DataWindow object is used in the w_profile window.

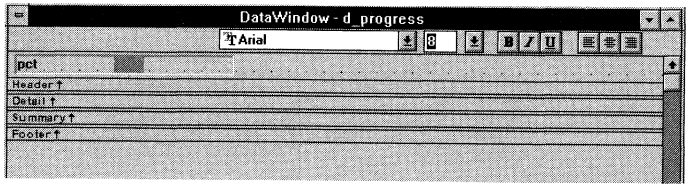
See also

w_profile

d_progress

Description

Graphically shows the progress of a process.



Library

UTLWIN.PBL

Usage

This DataWindow object is used in the w_progress window.

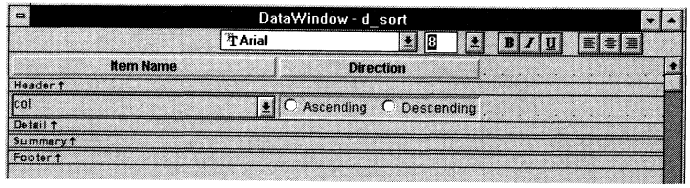
See also

w_progress

d_sort

Description

Allows the user to select the new sort order for another DataWindow.



Library

UTLWIN.PBL

Usage

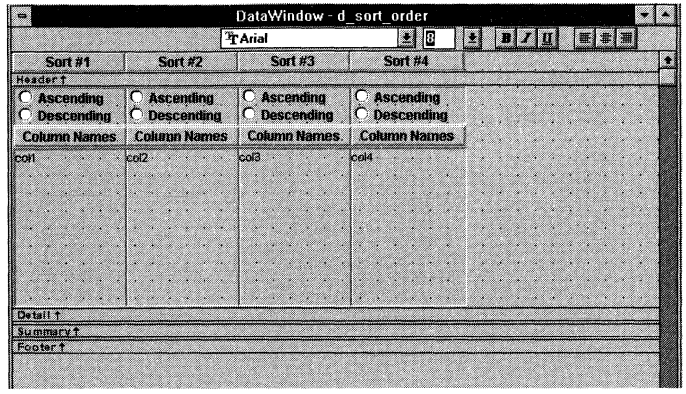
This DataWindow object is used in the w_sort window.

See also

w_sort

d_sort_order

Description Allows the user to select the new sort order for another DataWindow.



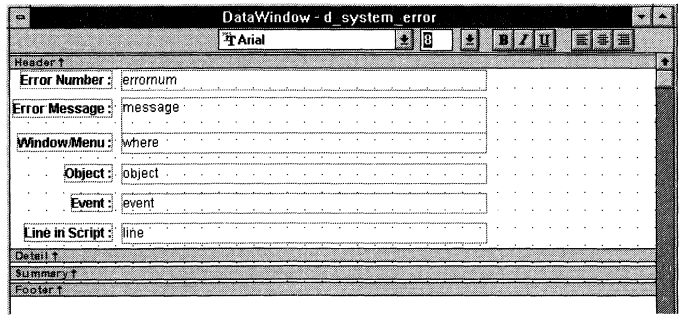
Library UTLWIN.PBL

Usage This DataWindow object is used in the w_sort_order window.

See also w_sort_order

d_system_error

Description Displays system error information.



Library UTLWIN.PBL

Usage This DataWindow object is used in the w_system_error window.

See also w_system_error

CHAPTER 6

Global Functions

About this chapter

This chapter describes the global functions in the Application Library. It includes descriptions and examples of each function. Functions are listed in alphabetical order.

f_app_open

Description

Opens the application's MDI frame window. It accesses the application INI file and is required for all applications that use the Application Library's application framework.

Library

SYS.PBL

Syntax

f_app_open (*appinifile, mainwindow, allowmultiples*)

Parameter	Description
appinifile	String containing the name of the application's INI file
mainwindow	Window variable of type w_sys_frame containing the name of the application's MDI frame window (the MDI frame window to be opened)
allowmultiples	Boolean value indicating whether multiple application instances are allowed

Return value

None

Usage

Call this function from the application object's Open event. The INI file provides database connection information. This is a sample INI file:

```
[Database1]
DBMS=ODBC
LogId=
LogPassword=
ServerName=
Database=HOTLINE
UserId=dba
DatabasePassword=
DbParm=Connectstring='DSN=HOTLINE;UID=DBA;PWD=SQL'
```

Use the PUBS.INI file as a template when creating your application's INI file.

Example

This example calls the f_app_open function in the Open event of the application. PUBS.INI is the INI file for the application and duplicate windows are not allowed.

```
f_app_open('PUBS.INI',w_frame,FALSE)
```

See also

w_sys_frame

f_block_text

Description

Receives a string and a block width for the string to be formatted and returns the string as a left-justified paragraph reformatted to the specified width.. It uses carriage return/line feed (~r~n) as the line separator.

Library

UTLFUNC.PBL

Syntax

f_block_text (*originalstring, blockwidth*)

Parameter	Description
originalstring	String containing the text to be formatted
blockwidth	Integer representing the width of the format block

Return value String. Returns the string formatted in the specified width.

Usage Use this global function to reformat text blocks to different lengths.

Example This statement passes the multiline edit text string mle_message.text to the f_block_text function. The string ls_err_msg will contain the MLE text formatted in a 60-character-wide, left justified block.

```
ls_err_msg = f_block_text (mle_message.text, 60)
```

f_boolean_to_string

Description Returns the passed boolean value as a string.

Library UTLFUNC.PBL

Syntax **f_boolean_to_string** (*booleanvalue*)

Parameters	Description
booleanvalue	Boolean value to be converted to a string.

Return value String. Returns "TRUE" for a TRUE boolean value and "FALSE" for a FALSE boolean value.

Usage Use this function to convert boolean values for display or printing as text.

Example This example sets the value in ls_last_record to "TRUE" if lb_last_record is TRUE. Otherwise, it sets the field to "FALSE".

```
Boolean lb_last_record
string ls_last_record
   .
   .
ls_last_record = f_boolean_to_string
(lb_last_record)
```

See also f_string_to_boolean

f_cascade_window

Description

Positions a window on opening so that it is just below the title or menu bar of the window that opened it.

> **Not for MDI applications**
> This global function is for non-MDI applications only.

Library

UTLFUNC.PBL

Syntax

f_cascade_window (*window*)

Parameter	Description
window	Window variable naming the popup, child, or response window to be opened. *Window* cannot specify a window of type Main, MDI, or MDI with MicroHelp.

Return value

None

Usage

Use this function in a non-MDI application to open cascaded windows. You should use this function with popup windows although it also works for child and response windows.

Example

This example is from the Open event of the cascading window.

```
f_cascade_window (this)
```

f_db_error

Description

Identifies whether a database error has occurred by checking the SQLCODE of the passed transaction object. If an error has occurred, f_db_error opens the window w_db_error, passing a string parameter containing the application-specific message and a boolean variable.

Window w_db_error displays the error message passed and the DBMS-specific error message. It also allows the user to print the error, close the error window, or continue using the application.

Library

UTLWIN.PBL

Syntax

f_db_error (*transactionobject, errormessage*)

Parameter	Description
transactionobject	Name of the programmer-specified transaction object currently being used
errormessage	Application-specific message that will be displayed above the database error message

Return value

Integer. Returns *0* if no database error is found and *1* if a database error occurred. If a database error occurs, then w_db_error displays.

Usage

Use this global function to trap errors after any embedded SQL is executed in scripts. Use the f_error_box function to check database errors for DataWindows.

Example

This example calls global function f_db_error and passes the default transaction object SQLCA with an application-specific error message.

```
IF f_db_error (SQLCA,"Error for Test Application") &
   <> 1 THEN
....
END IF
```

See also

f_debug_box
f_error_box
w_db_error
w_debug_box
w_error_box

f_dddw_lookup

Description

Positions an editable dropdown DataWindow based on what the user types.

Library

UTLFUNC.PBL

Syntax

f_dddw_lookup (*dwcontrol, dwcolumnname, dddwcolumndata, dddwcolumndescription*)

Parameter	Description
dwcontrol	DataWindow variable of the DataWindow control that contains the dropdown DataWindow
dwcolumnname	String specifying the name of the column displayed in the dropdown DataWindow
dddwcolumndata	String specifying which column of the dropdown DataWindow to use for data (typically the same as *dwcolumnname*)
dddwcolumndescription	String specifying which column of the dropdown DataWindow to use for the lookup

Return value

Boolean. Returns TRUE if the typed characters match an item in the dropdown DataWindow and FALSE if they do not.

Usage

Use this global function in the EditChanged and ItemChanged events to scroll the dropdown DataWindow based on what the user types. You can also use it to ensure that values exist in the dropdown DataWindow.

The dropdown DataWindow to be edited must be editable.

Example

This is an example of what you might code for the DataWindow control's EditChanged event. The code in this event, which occurs each time a key is pressed, first checks to see if the column is state_code. If the column is state_code, it calls the f_dddw_lookup function to position the dropdown DataWindow list, based on what the user typed.

```
string col

col = this.GetColumnName()
IF col = 'state_id' THEN
   f_dddw_lookup(this,col,'state_id','state_name')
END IF
```

This is an example of what you might code for the DataWindow control's ItemChanged event. It validates that the user-specified value in the state_code column exists in the dropdown DataWindow.

```
string col

col = GetColumnName()
IF col = 'state_id' THEN
   IF NOT f_dddw_lookup(this,col,'state_id', &
      'state_name') THEN
         SetActionCode(1)
         Return
   END IF
END IF
```

f_debug_box

Description

Opens the window w_debug_box, which displays the title and error message passed from the calling window.

Library

UTLWIN.PBL

Syntax

f_debug_box (*windowtitle*, *errormessage*)

Parameter	Description
windowtitle	String containing a name for the w_debug_box window title. Usually passes the current window title (that is, this.title or parent.title).
errormessage	String variable containing the error message to be displayed.

Return value

None

Usage

Use the f_debug_box function to communicate variables and status information when debugging applications.

The w_debug_box window is nonmodal, which means that it can remain open while the user performs other actions associated with the application, and it can display multiple messages. Multiple calls to the f_debug_box function append the new text to the existing window text.

Example

This example opens window w_debug_box if the PrintOpen function fails.

```
IF PrintOpen() = -1 THEN
   f_debug_box (this.title, "Error in Print event")
END IF
```

See also

f_db_error
f_error_box
w_db_error
w_debug_box
w_error_box

f_display_file

Description

Displays the specified text file in the w_file_display window.

Library

UTLFUNC.PBL

Syntax

f_display_file (*filename*)

Parameter	Description
filename	String containing the fully qualified name of the file to be displayed in the w_display_file window. *Filename* must use the .TXT extension

Return value

None

Usage

Use this function to display text files in the w_file_display window.

Example

This example displays the file if the GetFileOpenName function succeeds.

```
int result
string docname, named
result = GetFileOpenName("Select File", docname, &
   named, "txt", "Text Files, *.txt")
IF result = 1 THEN f_display_file (docname)
```

See also

w_file_display

f_dw_fill_ddlb

Description

Fills the items in a DataWindow dropdown listbox from a specified database table.

Library

UTLFUNC.PBL

Syntax

f_dw_fill_ddlb (*datawindowname, sqlstatement, colname*)

Parameter	Description
datawindowname	DataWindow variable pointing to the DataWindow containing the dropdown listbox
sqlstatement	String containing the SQL statement used to fill the dropdown listbox
colname	String containing the name of the column to be populated

Return value

Boolean. Returns TRUE if the SQL statement succeeds and FALSE if it does not.

Usage

This function is provided to maintain backward compatibility. For new applications, use dropdown DataWindows instead.

Example

This example fills the dropdown listbox column STATE in dw_location_info with values from the table STATES.

```
f_dw_fill_ddlb(dw_location_info, &
   "SELECT state, state_code FROM STATES", "STATE")
```

f_dw_get_attributes

Description

Provides information about a series of attributes for a single object in a DataWindow.

Library

UTLFUNC.PBL

Syntax

f_dw_get_attributes (*dwcontrol, dwobject, attributelist*)

Parameter	Description
dwcontrol	DataWindow variable pointing to the DataWindow control containing the attributes to be returned
dwobject	String naming the object on the DataWindow whose attributes are to be returned
attributelist	String containing a comma-separated list of attributes to be returned

Return value

String containing a comma-separated list of attribute names. Returned attributes are separated by newline (~n) characters.

Usage

Use this function to get information about a series of attributes for a single object on a DataWindow.

You can use the f_get_token function to separate the returned attribute values.

Example

This example calls function f_dw_get_attributes to get information about the object au_id.

```
string results
integer li_x, li_y
boolean lb_visible

results = &
   f_dw_get_attributes (dw_1,'au_id','x,y,visible')

li_x = integer(f_get_token(results,'~n'))
li_y = integer(f_get_token(results,'~n'))
lb_visible = f_string_to_boolean(results)
```

See also

f_dw_get_objects
f_dw_get_objects_attrib
f_dw_set_color
f_dw_set_color_row
f_get_token

f_dw_get_objects

Description

Parses the list of objects contained in the data object associated with a DataWindow control, placing their names into a string array passed by reference and returning the number of names in the array. You can control the objects returned by type and by band.

For information on valid band and object type names, see the PowerBuilder *Function Reference*.

Library

UTLFUNC.PBL

Syntax

f_dw_get_objects (*dwcontrol, objectlist, objecttype, band*)

Parameter	Description
dwcontrol	DataWindow variable pointing to the DataWindow control containing the objects to be returned.
objectlist	Unbounded string array into which f_dw_get_objects places the data object names. This array is passed by reference.
objecttype	String indicating the object type to be searched on. Specifying an asterisk (*) returns the names of all objects.
band	String indicating which band to search for *objecttype*. Specifying an asterisk (*) returns data object names for all bands.

Return value

Integer indicating the number of names in the *objectlist* array.

Usage

Use this function to get information about the DataWindow object associated with a window's DataWindow control.

> **All DataWindow columns must have names**
> If this function will be used against columns, all of the DataWindow's columns must have names.

Examples

This example returns the names of all text objects in the header band of dw_1 into the mylist array; the number of names is returned in obj_num.

```
integer  obj_num
string   mylist[ ]
```

```
obj_num = f_dw_get_objects(dw_1, mylist, &
   "text", "header")
```

This example returns the names of all column objects in dw_1 into the mylist array; the number of names is returned in obj_num.

```
integer  obj_num
string   mylist[ ]

obj_num = f_dw_get_objects(dw_1,mylist,"column","*")
```

This example returns the names of all objects in the summary band of dw_1 into the mylist array; the number of names is returned in obj_num.

```
integer  obj_num
string   mylist[ ]

obj_num = f_dw_get_objects(dw_1, mylist, &
   "*", "summary")
```

This example returns the names of all objects in dw_1 into the mylist array; the number of names is returned in obj_num.

```
integer  obj_num
string   mylist[ ]

obj_num = f_dw_get_objects(dw_1, mylist, "*", "*" )
```

See also

f_dw_get_attributes
f_dw_get_objects_attrib
f_dw_set_color
f_dw_set_color_row

f_dw_get_objects_attrib

Description

Parses the list of objects contained in the data object associated with a DataWindow control, placing their names into a string array passed by reference and returning the number of names in the array. You can control the objects returned by type and by band. Additionally, you can request attribute values by passing a comma-separated list of attribute names. If you request attributes, the function returns them as part of the array, separated from the object name by newline (~n) characters.

For information on valid band and object type names, see the PowerBuilder *Function Reference*.

Library

UTLFUNC.PBL

Syntax

f_dw_get_objects_attrib (*dwcontrol, objectlist, objecttype, band, attributes*)

Parameter	Description
dwcontrol	DataWindow variable pointing to the DataWindow control containing the objects to be returned.
objectlist	Unbounded string array into which f_dw_get_objects_attrib places the data object names. This array is passed by reference.
objecttype	String indicating the object type to be searched on. Specifying an asterisk (*) returns names and attributes of all objects.
band	String indicating which band to search for *objecttype*. Specifying an asterisk (*) returns data object names and attributes for all bands.
attributes	String containing a comma-separated list of attributes.

Return value

Integer indicating the number of names in the *objectlist* array.

Usage

Use this function to get information about the DataWindow object and object attributes associated with a window's DataWindow control.

> **All DataWindow columns must have names**
> If this function will be used against columns, all of your columns must have names.

Examples

This example returns the names of all text objects in the header band of dw_1 into the mylist array, and the number of names is returned into obj_num. Their x and y coordinates are also returned.

```
integer  obj_num
string   mylist[ ]

obj_num = f_dw_get_objects_attrib(dw_1, mylist, &
   "text","header","x,y")
```

This example returns the names of all column objects in dw_1 into the mylist array, and the number of names is returned into obj_num. No attributes are requested.

```
integer  obj_num
string   mylist[ ]

obj_num = f_dw_get_objects_attrib(dw_1, mylist, &
   "column","*","")
```

This example returns the names of all objects in the summary band of dw_1 into the mylist array, and the number of names is returned into obj_num. Information about the object's height is also requested.

```
integer  obj_num
string   mylist[ ]

obj_num = f_dw_get_objects_attrib(dw_1, mylist, &
   "*","summary","height")
```

This example returns the names of dw_1 objects into the mylist array, and the number of names is returned into obj_num. No attributes are requested.

```
integer  obj_num
string   mylist[ ]

obj_num = f_dw_get_objects_attrib(dw_1, mylist, &
   "*", "*" ,"")
```

See also

f_dw_get_attributes
f_dw_get_objects
f_dw_set_color
f_dw_set_color_row
f_get_token

f_dw_getcolnames

Description

Gets the names for all columns in a DataWindow.

Library

UTLFUNC.PBL

Syntax

f_dw_getcolnames (*dwcontrol, colnames*)

Parameter	Description
dwcontrol	DataWindow variable of the DataWindow control containing the columns to be returned.

Parameter	Description
colnames	Unbounded string array into which f_dw_getcolnames places the column names. This array is passed by reference.

Return value

Integer indicating the number of names in the *colname* array.

Usage

Use this function to determine the names of all columns in a DataWindow.

Example

This example returns all DataWindow column names into the ls_colnames array. The li_colnum variable indicates the number of entries in the ls_colnames array.

```
integer  li_colnum
string   ls_colnames[]

li_colnum = f_dw_getcolnames (dw_sheet, ls_colnames)
```

See also

f_dw_get_attributes
f_dw_get_objects
f_dw_get_objects_attrib
f_dw_getheaderlabel
f_dw_getvisiblecolumns
f_dwobjectatpointer

f_dw_getheaderlabel

Description

Determines an appropriate header for a column. This function first looks for a static text label with column name plus the _t suffix. If that isn't found, it then looks for a tag value. If there is no tag value, the function returns the column name. And if the column name isn't available, it returns an empty string.

Library

UTLFUNC.PBL

Syntax

f_dw_getheaderlabel (*dwcontrol, colname*)

Parameter	Description
dwcontrol	DataWindow control variable of the DataWindow control containing the column
colname	String specifying the column name for which a heading value will be returned

Return value

String containing the column label. If a heading cannot be found it returns an empty string and displays an error message.

Usage

Use this function when you want to use a column's header text instead of the column name in a message box.

Example

This example returns the column header for emp_id into ls_colheader.

```
string   ls_colheader

ls_colheader =
f_dw_getheaderlabel(dw_sheet,"emp_id")
```

See also

f_dw_get_attributes
f_dw_get_objects
f_dw_get_objects_attrib
f_dw_getcolnames
f_dw_getvisiblecolumns
f_dwobjectatpointer

f_dw_getvisiblecolumns

Description

Gets the names of all visible columns in a DataWindow.

Library

UTLFUNC.PBL

Syntax

f_dw_getvisiblecolumns (*dwcontrol, colnames*)

Parameter	Description
dwcontrol	DataWindow variable of the DataWindow control containing the columns to be returned.

Parameter	Description
colnames	Unbounded string array into which f_dw_getvisiblecolumns places the column names. This array is passed by reference.

Return value

Integer indicating the number of columns in the *colname* array.

Usage

Use this function to determine the names of all visible columns in a DataWindow.

Example

This example returns the names of all visible DataWindow columns into the ls_colnames array. The li_colnum variable indicates the number of entries in the ls_colnames array.

```
integer  li_colnum
string   ls_colnames[]

li_colnum = &
   f_dw_getvisiblecolumns (dw_sheet,ls_colnames)
```

See also

f_dw_get_attributes
f_dw_get_objects
f_dw_get_objects_attrib
f_dw_getcolnames
f_dw_getheaderlabel
f_dwobjectatpointer

f_dw_objectatpointer

Description

Separates the string returned by the PowerBuilder GetObjectAtPointer function into object name and row number.

Library

UTLFUNC.PBL

Syntax

f_dw_objectatpointer (*dwcontrol, rownumber, objectname*)

Parameter	Description
dwcontrol	DataWindow variable of the DataWindow control containing the object under the pointer.

Parameter	Description
rownumber	Integer into which f_dwobjectatpointer places the row number. This integer is passed by reference.
objectname	String into which f_dwobjectatpointer places the object name. This string is passed by reference.

Return value

Integer. Returns *1* if the function succeeded and *-1* if it did not.

Usage

The PowerBuilder GetObjectAtPointer function returns a string containing the name of the object under the pointer, a nondisplaying tab, and the row number. Use the f_dwobjectatpointer function to separate this string into separate variables.

Example

This example returns the row number into li_row and the object name into ls_object.

```
integer  li_row, li_return
string   ls_object, ls_obj_and_row

ls_obj_and_row = dw_sheet.GetObjectAtPointer()
li_return = &
   f_dw_objectatpointer(dw_sheet,li_row,ls_object)
```

See also

f_dw_get_attributes
f_dw_get_objects
f_dw_get_objects_attrib
f_dw_getcolnames
f_dw_getheaderlabel
f_dw_getvisiblecolumns

f_dw_print

Description

Prints the specified DataWindow to a printer or a file, optionally displaying the w_dw_print_options window to prompt the user for more information.

Library

UTLWIN.PBL

Syntax

f_dw_print (*dwcontrol, numberofcopies, filename*)

Parameter	Description
dwcontrol	DataWindow variable of the DataWindow control whose contents will be printed.
numberofcopies	Integer specifying the number of copies to be printed.
filename	String containing the fully qualified name of the file to contain the DataWindow contents. PowerBuilder creates this file in the format of the current printer driver (for example, PCL or PostScript).

Return value

Integer. Returns *1* if the function succeeded and *-1* if it did not (or if the user pressed Cancel in the w_dw_print_options window).

Usage

Use this function to control DataWindow printing and print options.

To display the w_dw_print_options window, *numberofcopies* must be 0 (zero) and *filename* must contain an empty string. If *numberofcopies* is greater than 0 or if *filename* contains a nonempty string, the function does not display the w_dw_print_options window but instead prints the DataWindow immediately (to either a printer or a file, depending on whether *filename* is empty).

Example

This example prints the dw_sheet DataWindow and displays the w_dw_print_options window to prompt the user for additional information. If the user presses Cancel or the print function fails, then an error message is displayed.

```
string   ls_filename = ""
integer  li_num_copies = 0
integer  li_return

li_return = f_dw_print (dw_sheet, &
   li_num_copies, ls_filename)
IF li_return = -1 THEN
   MessageBox("Print Error", &
      "User pressed Cancel or an error occured")
END IF
```

See also

f_print_file
w_dw_print_options

f_dw_set_color

Description

Sets values in the specified column to the color designated in the parameter *colortrue* if the expression is evaluated to be TRUE. Otherwise, the column value is set to the color designated in the parameter *colorfalse*.

For a list of color values, see the PowerBuilder *Function Reference*.

Library

UTLFUNC.PBL

Syntax

f_dw_set_color (*dwcontrol, column, expression, colortrue, colorfalse*)

Parameter	Description
dwcontrol	DataWindow variable pointing to the DataWindow control containing the column whose color will be modified
column	String containing the name of the database column whose color will be modified
expression	String expression to be evaluated
colortrue	Long representing the color used if *expression* is TRUE
colorfalse	Long representing the color used if *expression* is FALSE

Return value

None

Usage

Use this function to selectively highlight values in a DataWindow.

> **Use the RGB function**
> Use the PowerBuilder RGB function to name the color.

Example

This example changes the color of column PRICE in dw_1 from black to red when its value is greater than 100.

```
f_dw_set_color(dw_1, "PRICE", "PRICE > 100", &
   RGB(255,0,0),RGB(0,0,0))
```

See also

f_dw_set_color_row

f_dw_set_color_row

Description

Sets all columns on the row to the color designated in the *colortrue* parameter if the passed expression is TRUE. Otherwise, the row color is set to the value in *colorfalse*.

For a list of color values, refer to the PowerBuilder *Function Reference*.

Library

UTLFUNC.PBL

Syntax

f_dw_set_color_row (*dwcontrol*, *expression*, *colortrue*, *colorfalse*)

Parameter	Description
dwcontrol	DataWindow variable pointing to the DataWindow control containing the column whose color will be modified
expression	String expression to be evaluated
colortrue	Long representing the color used if *expression* is TRUE
colorfalse	Long representing the color used if *expression* is FALSE

Return value

None

Usage

Use this function to selectively highlight rows in a DataWindow.

> **Use the RGB function**
> Use the PowerBuilder RGB function to name the color.

Example

This example sets the row to red if the field ACTIVITY is equal to SALE; otherwise, the color is set to green.

```
f_dw_set_color_row (dw_1, "IF ACTIVITY = 'SALE'", &
   RGB(255,0,0), RGB(0,255,0))
```

See also

f_dw_set_color

f_error_box

Description

Opens the window w_error_box, which displays the title and error message passed from f_error_box.

The w_error_box window is non-modal window, which means that it can remain open while the user performs other actions associated with the application.

Library

UTLWIN.PBL

Syntax

f_error_box (*windowtitle*, *errormessage*)

Parameter	Description
windowtitle	String containing w_error_box window title. Usually contains the title of the window in which the error occurred.
errormessage	String containing the error message to be displayed.

Return value

None

Usage

This global function is typically called from the DBError event of a DataWindow control.

For an example, see the DBError event for the uo_dw DataWindow user object found in SYS.PBL.

Example

This example opens window w_error_box in the Clicked event of a print commandbutton if the PrintOpen function fails.

```
IF PrintOpen( ) = -1 THEN
   f_error_box(parent.title, &
      "Error in clicked event of object cb_print")
END IF
```

See also

f_db_error
f_debug_box
w_db_error
w_debug_box
w_error_box

f_exit_status

Description

Opens the window w_exit_status, which allows the user to choose the action to be taken if the modified data has not been saved.

Library

UTLWIN.PBL

Syntax

f_exit_status (*windowtitle, saveormove*)

Parameter	Description
windowtitle	String specifying w_exit_status window title. Usually contains the title of the window to be closed.
saveormove	String indicating whether data will be lost due to closing a window or moving from a DataWindow row in which data has been changed for the row: ♦ **S** – Unsaved changes will be lost because the window is closing ♦ **M** – Unsaved changes will be lost because the user is moving from a DataWindow row with unsaved changes

Return value

One of the following single-character string values: S – save data and then close the window; E – close the window without saving; C – Cancel the request.

Usage

In a window's CloseQuery event, check to see if any changed data has not been saved to the database. If there is unsaved data, use this function to let the user determine how to proceed.

Example

This example checks to see if data has been modified on the window and not saved. If TRUE, use the f_exit_status function to open w_exit_status and prompt the user to determine if information should be saved.

```
string ls_status

IF ModifiedCount(dw_1) <> 0 THEN
   ls_status = f_exit_status(this.title,"S")
END IF
CHOOSE CASE ls_status
   CASE "S"
      // Save data and close the window.
   CASE "E"
      // Close the window without saving.
```

```
        CASE "C"
           // Cancel request; return to window.
    END CHOOSE
```

See also w_exit_status

f_get_parm

Description

Returns the value that corresponds to the passed variable name.

The main purpose of this window (to pass parameters when opening and closing windows) has been replaced with the functionality provided by OpenWithParm and CloseWithReturn functions. Global function f_get_parm has been retained for backward compatibility.

Library

UTLFUNC.PBL

Syntax

f_get_parm (*variable*)

Parameter	Description
variable	String whose value is to be returned

Return value

String. If nothing is found, it returns NULL.

Usage

This function, which is provided for backward compatibility, can be used in place of global variables when communicating between objects. New applications should use OpenWithParm and CloseWithReturn instead.

Example

This example retrieves a DataWindow based on the value stored in the string instance variable is_state_name.

```
Retrieve(dw_location_info, &
   f_get_parm (is_state_name))
```

See also

f_pop_parm
f_push_parm
f_set_parm
w_hold_parms

f_get_string

Description

Opens the response window w_get_string, which allows the user to enter the requested information. It returns a string containing the information the user entered on window w_get_string.

Library

UTLWIN.PBL

Syntax

f_get_string (*windowtitle, maxlength, caseindicator, currentvalue*)

Parameter	Description
windowtitle	String indicating the title to be used in w_get_string.
maxlength	Integer specifying the maximum number of characters that can be entered.
caseindicator	Single-character string indicating the case of the string to be entered: U – Uppercase, L – Lowercase, A – Any case.
currentvalue	Optional string containing the current value of what is to be modified.

Return value

String

Usage

Use this function to prompt the user for a string value.

Examples

This example prompts the user for a string 40 characters long and uppercase only. The title of w_get_string will be equal to the title of the calling window.

```
string   ls_string

ls_string = &
   f_get_string (parent.title,40,"U",ls_string)
```

This example prompts the user for multiple strings different in length and case type. The title of the new window opened is equal to the information being requested.

```
string   ls_custnum, ls_dept, ls_custname

ls_custnum = &
   f_get_string("Customer number",8,"U",ls_custnum)
ls_dept = f_get_string("Department", 4, "U",ls_dept)
```

```
ls_custname = &
   f_get_string("Customer name", 40,
"A",ls_custname)
```

See also w_get_string

f_get_token

Description

Returns the token from a passed string. This function receives as arguments a string from which the token is to be removed (from the left) and the separator character.

> **What is a token?**
> A token is a collection of characters separated by a delimiter.

If the separator character appears in the string, the function returns the token, not including the separator character. If the separator character does not appear in the string, the function returns the entire string. In either case, the source string is truncated on the left by the length of the token and separator character, if any.

Library UTLFUNC.PBL

Syntax **f_get_token** (*sourcestring, separatorstring*)

Parameter	Description
sourcestring	String to be parsed. It is passed by reference.
separatorstring	String containing the separator character(s).

Return value String. Returns the token if *separatorstring* is found within *sourcestring* and returns the entire *sourcestring* if *separatorstring* is not found.

Usage Use this function to break up strings with embedded values into separate strings.

Example

This example calls function f_get_token to break out the information returned from f_dw_get_attributes.

```
string results
integer li_x, li_y
boolean lb_visible

results = &
   f_dw_get_attributes(dw_1,'au_id','x,y,visible')

li_x = integer(f_get_token(results,'~n'))
li_y = integer(f_get_token(results,'~n'))
lb_visible = f_string_to_boolean(results)
```

See also

f_dw_get_attributes

f_global_replace

Description

Replaces all occurrences of a string in one string with another string.

Library

UTLFUNC.PBL

Syntax

f_global_replace (*source, lookfor, replacewith*)

Parameter	Description
source	String to be searched
lookfor	String to be found
replacewith	String used to replace *lookfor*

Return value

String that includes the substituted value.

Usage

Use this function to substitute one value for another in a string.

Example

This example replaces all occurrences of red in string1 with green.

```
string results, string1
   .
   .
results = f_global_replace (string1,'red','green')
```

f_invert_color

Description

Returns the inverse of the color passed.

Library

UTLFUNC.PBL

Syntax

f_invert_color (*colorvalue*)

Parameter	Description
colorvalue	Long representing the value of the color whose inverse is returned by the function

For a list of color values, refer to the PowerBuilder *Function Reference.*

Return value

Long specifying the inverse color.

Usage

Use this function to determine the inverse of a passed color.

> **Use the RGB function**
> Use the PowerBuilder RGB function to name the original color.

Example

This example sets the column to the inverse of the current color, green, when the local field li_price is larger than 1000.

```
long ll_inverse_color, ll_current_color

ll_current_color = RGB (0,128, 0)
ll_inverse_color = f_invert_color (ll_current_color)

f_dw_set_color(dw_1, "PRICE", "PRICE > 1000", &
   ll_inverse_color, ll_current_color)
```

See also

f_dw_set_color
f_dw_set_color_row

f_julian

Description

Converts a date from Date format (for example, 1994-06-29) into julian format (for example, 1994-180)

Library

UTLFUNC.PBL

Syntax

f_julian (*date, julianyear, julianday*)

Parameter	Description
date	Date from which julian date is to be calculated.
julianyear	Integer into which f_julian returns the year portion of the julian date. This value is passed by reference.
julianday	Integer into which f_julian returns the day number. This value is passed by reference.

Return value

String containing *year - day*, for example, *1994 - 230*.

Usage

Use this function to calculate a julian date (such as, 1994-180) from a date in Date format (such as, 1994-06-29).

Example

This example calculates the julian date for June 29, 1994. It returns *1994-180* into the ls_julian_all field; *1994*, into the li_julian_year field; and *180* into the li_julian_day field.

```
string   ls_julian_all
integer  li_julian_year, li_julian_day
date  ld_date_in

ld_date_in = Date("1994-06-29")

ls_julian_all = f_julian(ld_date_in, &
   li_julian_year,li_julian_day)
```

f_login

Description

Opens window w_login, which prompts the user to log in to the application. The user must supply a user ID and password. The global function passes an INI file (usually the application's) as a parameter to w_login and is typically used in the application or main window's Open event.

Library

UTLWIN.PBL

Syntax

f_login (*inifilename*)

Parameter	Description
inifilename	The path and complete name of the INI filename associated with the current application. The INI file contains specific DBMS information used for connecting to the database.

Return value

Boolean. Returns TRUE if the login succeeded and FALSE if it did not.

Usage

Use this function for database login. The INI file provides database connection information. A sample INI file is shown below:

```
;Setup for SQLCA
[Database1]
DBMS=ODBC
LogId=
LogPassword=
ServerName=
Database=HOTLINE
UserId=dba
DatabasePassword=
DbParm=Connectstring='DSN=HOTLINE;UID=DBA;PWD=SQL'
```

Example

This example call the f_login function in the Open event of the frame window. If the user is unable to log in, the application closes.

```
IF NOT f_login("c:\pubs\pubs.INI") THEN
   halt close
END IF
```

See also

w_login
w_sys_frame

f_lookupcode

Description

Returns the code value associated with the passed display value for a dropdown DataWindow column.

Library

UTLFUNC.PBL

Syntax

f_lookupcode (*dwcontrol, displayvalue, columnname*)

Parameter	Description
dwcontrol	DataWindow variable of the DataWindow control containing the column.
displayvalue	String containing the display value
columnname	String containing the name of the column for *displayvalue*

Return value

String containing the code value that corresponds to *displayvalue*.

Usage

Use this function for two purposes:

- To determine the code value (typically the value that is actually stored in the database) that corresponds to a dropdown DataWindow value without scrolling the dropdown DataWindow.
- To ensure that a user-specified value has a corresponding code in the dropdown DataWindow.

Example

This example returns the 2-character state code that corresponds to the ls_state value.

```
string   ls_code
string   ls_state = "Minnesota"

ls_code = f_lookupcode ( dw_sheet,ls_state,"state" )
```

See also

f_lookupdisplay

f_lookupdisplay

Description

Returns the current display value associated a particular row and column. This function works with dropdown DataWindows, dropdown listboxes, and radiobuttons.

Library

UTLFUNC.PBL

Syntax

f_lookupdisplay (*dwcontrol, rownumber, columnname*)

Parameter	Description
dwcontrol	DataWindow variable of the DataWindow control containing the column
rownumber	Integer indicating the row that contains the code value
columnname	String indicating the column that contains the code value

Return value

String containing the display value for the code specified by *rownumber* and *columnname*.

Usage

Use this function to determine the display value for a particular row and column.

Example

This example returns the display value for the 2-character state code.

```
string   ls_display
long     ll_rownum

ll_rownum = dw_sheet.GetRow()
ls_display = &
   f_lookupdisplay ( dw_sheet, ll_rownum, "state")
```

See also

f_lookupcode

f_maillogoff

Description

Ends a mail session.

Library

UTLFUNC.PBL

Syntax

f_maillogoff (*mailsession*)

Parameter	Description
mailsession	MailSession variable identifying the session from which you want to log off

Return value

None

Usage

Use this function to end an e-mail session.

Example

This example ends the ims_sess (instance variable) e-mail session.

```
f_maillogoff ( ims_sess )
```

See also

f_maillogon
f_mailsend
f_mailsendnoaddress

f_maillogon

Description

Starts a mail session.

Library

UTLFUNC.PBL

Syntax

f_maillogon (*mailsession*)

Parameter	Description
mailsession	MailSession variable identifying the session you want to log on to. This variable is passed by reference.

Return value

Boolean. Returns TRUE if the function succeeds and FALSE if it does not.

Usage

Use this function to establish a new mail session.

Example

This example establishes a new mail session, storing the MailSession in the ims_sess instance variable.

```
IF NOT f_maillogon (ims_sess) THEN
   MessageBox("Logon Failed","E-mail logon failed")
END IF
```

See also

f_maillogoff
f_mailsend
f_mailsendnoaddress

f_mailsend

Description

Sends mail to the specified recipients. This function will create a mail session automatically if no current mail session exists.

Library

UTLFUNC.PBL

Syntax

f_mailsend (*recipients*, cc, *mailsession*, *subject*, *notetext*, *filename*, *pathname*, *mailmessage*, *returnreceipt*, *framewindow*)

Parameter	Description
recipients	Unbounded string array naming message recipients. The names in this array must be valid names within your e-mail system.
cc	String specifying either yes (display CC dialog box) or no (don't display the CC dialog box)
mailsession	MailSession variable identifying the session you want to use. This variable is passed by reference.
subject	String containing the message subject.
notetext	String containing the body of the message.
filename	Unbounded string array whose elements specify the names of files attached to the message.

Parameter	Description
pathname	Unbounded string array whose elements specify the corresponding path names of files attached to the message. For example, the first element in the *pathname* array specifies the path for the first element in the *filename* array, and so on.
mailmessage	MailMessage variable that will be used by the function. This variable is passed by reference.
returnreceipt	Boolean specifying whether you want confirmation that the message was received.
framewindow	Window variable specifying the frame window.

Return value

Boolean. Returns TRUE if the function succeeds and FALSE if it does not.

Usage

Use this function to send a message.

> **PowerBuilder supports MAPI-compliant e-mail systems**
> PowerBuilder supports MAPI (messaging application program interface). You can use the Application Library's e-mail functions with an MAPI-compliant electronic mail system. Microsoft Mail is an example of a MAPI-compliant electronic mail system.

Example

This example sends a mail message for the mail session contained in the ims_sess instance variable.

```
mailMessage    lmm_message
string   ls_to[], ls_subject, ls_text, ls_files[]
string   ls_path[], ls_cc
boolean  lb_returnreceipt, lb_return
window   lw_frame

lw_frame = This.ParentWindow()
ls_subject = This.sle_subject
ls_text = This.mle_messagetext
lb_returnreceipt = This.cbx_returnreceipt.Checked

... // Logic to access recipients goes here.
... // Populates ls_to[] array

... // Logic to access specified files and
... // pathnames goes here.
... // Populates ls_files[] and ls_path[] arrays.

IF NOT f_mailsend ( ls_to, ls_cc, ims_sess, &
   ls_subject, ls_text, ls_files, ls_path, &
   lmm_message, lb_returnreceipt, lw_frame ) THEN
```

```
            MessageBox("Failure","Couldn't send message.")
    END IF
```

See also

f_maillogon
f_maillogoff
f_mailsendnoaddress

f_mailsendnoaddress

Description

Sends mail to unknown recipients. This function results in the display of the e-mail system's address book, which prompts the user for the names of recipients. It will also create a mail session automatically if no current mail session exists.

Library

UTLFUNC.PBL

Syntax

f_mailsendnoaddress (*mailsession*, *subject*, *notetext*, *filename*, *pathname*, *mailmessage*, *returnreceipt*, *framewindow*)

Parameter	Description
mailsession	MailSession variable identifying the session you want to use. This variable is passed by reference.
subject	String containing the message subject.
notetext	String containing the body of the message.
filename	Unbounded string array whose elements specify the names of files attached to the message.
pathname	Unbounded string array whose elements specify the corresponding path names of files attached to the message. For example, the first element in the *pathname* array specifies the path for the first element in the *filename* array, and so on.
mailmessage	MailMessage variable that will be used by the function. This variable is passed by reference.
returnreceipt	Boolean specifying whether you want confirmation that the message was received.
framewindow	Window variable specifying the frame window.

Return value

Boolean. Returns TRUE if the function succeeds and FALSE if it does not.

Usage

Use this function to send mail and allow the user to specify recipients through the e-mail system's address book.

Example

This example sends a mail message for the mail session contained in the ims_sess instance variable.

```
mailMessage    lmm_message
string   ls_subject, ls_text, ls_files[]
string   ls_path[]
boolean  lb_returnreceipt, lb_return
window   lw_frame

lw_frame = This.ParentWindow()
ls_subject = This.sle_subject
ls_text = This.mle_messagetext
lb_returnreceipt = This.cbx_returnreceipt.Checked

// Logic to access specified files and
// pathnames goes here.
// Populates ls_files[] and ls_path[] arrays.
.
.
.
IF NOT f_mailsendnoaddress (ims_sess, ls_subject, &
   ls_text, ls_files, ls_path, lmm_message, &
   lb_returnreceipt, lw_frame ) THEN
      MessageBox("Failure","Couldn't send message.")
END IF
```

See also

f_maillogon
f_maillogoff
f_mailsend

f_parsedisplaydata

Description

Separates the display~*t*code values of an EditMask control or RadioButton-style column.

Library

UTLFUNC.PBL

Syntax

f_parsedisplaydata (*initialstring, displaycodearray*)

Parameter	Description
initialstring	String containing the value to be parsed.
displaycodearray	Two-dimensional string array to contain the separated values. The bounds of this array must be [100,2]. This variable is passed by reference.

Return value

Integer indicating the number of rows in *displaycodearray*.

Usage

Use this function to separate tab-delimited strings.

Example

This example uses the f_parsedisplaydata function to separate the values in the code table for the status column.

```
integer  li_rows
string   ls_original, ls_displayvalues[100,2]

is_original = dw_1.Describe("status.values")
li_rows = &
   f_parsedisplaydata(ls_original,ls_displayvalues)
```

See also

f_parseleftright
f_parsestringintoarray

f_parseleftright

Description

Returns everything to the left of the delimiter into one string and everything to the right of the delimiter into another string.

Library

UTLFUNC.PBL

Syntax

f_parseleftright (*initialstring, delimiter, leftstring, rightstring*)

Parameter	Description
initialstring	String containing the value to be parsed.

Parameter	Description
delimiter	String containing the delimiter used to separate *initialstring*.
leftstring	String to contain the characters to the left of *delimiter*. This variable is passed by reference.
rightstring	String to contain the characters to the right of *delimiter*. This variable is passed by reference.

Return value

Integer. Returns *1* if delimiter was found in *initialstring* and *-1* if it was not.

Usage

Use this function to separate a two-part string into two separate strings.

Example

This example separates the result of a GetObjectAtPointer function.

```
integer  li_return
string   ls_original, ls_left, ls_right
string   ls_delimiter = "~t"

ls_original = dw_sheet.GetObjectAtPointer()
li_return = &
   f_parseleftright(ls_original, li_delimiter, &
      ls_left, ls_right)
```

See also

f_parsedisplaydata
f_parsestringintoarray

f_parsestringintoarray

Description

Separates a string into an array, based on the passed delimiter.

Library

UTLFUNC.PBL

Syntax

f_parsestringintoarray (*initialstring, delimiter, stringarray*)

Parameter	Description
initialstring	String containing the value to be parsed.
delimiter	String containing the delimiter used to separate *initialstring*.

Parameter	Description
stringarray	Unbounded string array to contain the separated values. This variable is passed by reference.

Return value

Integer indicating the number of rows in *stringarray*.

Usage

Use this function to convert output from the Describe function into elements in an array.

Example

This example separates the string returned by a Describe function.

```
integer  li_num_rows
string   ls_original, ls_delimiter = "~t"
string   ls_elements[]

ls_original =
dw_sheet.Describe("DataWindow.Objects")
li_num_rows = &
   f_parsestringintoarray(ls_original,ls_delimiter &
      ls_elements )
```

See also

f_parsedisplaydata
f_parseleftright

f_pop_parm

Description

Removes a value from the parameter stack that is contained in the window w_hold_parms. You insert values into the stack array using the f_push_parm function.

The main purpose of this window (to access passed parameters when opening windows) has been replaced with the functionality provided by OpenWithParm and CloseWithReturn functions. Global function f_pop_parm has been retained for backward compatibility.

Library

UTLFUNC.PBL

Syntax

f_pop_parm ( )

Return value

String value of the element on the top of the stack.

Usage

This function is provided for backward compatibility. New applications should use the OpenWithParm and CloseWithReturn functions instead.

> **Note**
> To use this function, you must open the w_hold_parms window in the application's Open event.

Example

This example uses one window to select parameters needed for retrieving information. These values are pushed on the parameter stack so that the window that will display the detail information can have access to the parameters needed for the retrieval.

The selection window pushes the values for li_ref_num, li_product_id, ls_activity, ld_date, and ls_state_code onto the parameter stack. These values are needed for the retrieval of the DataWindow dw_1, which is on the detail information window.

Script in Selection Window

```
f_push_parm (string(li_ref_num))
f_push_parm (string( li_product_id ))
f_push_parm (ls_activity )
f_push_parm (string(ld_date, mm-dd-y ))
Open(w_information)
```

Script in the w_information Window

```
li_new_date = date (f_pop_parm ())
li_new_activity = f_pop_parm ( )
li_new_product_id  = integer (f_pop_parm ())
li_new_ref_num = integer(f_pop_parm ())
Retrieve(dw_1, li_new_ref_num, li_new_product_id, &
   li_new_activity, li_new_date)
```

See also

f_get_parm
f_push_parm
f_set_parm
w_hold_parms

f_print_file

Description

Sends the specified file to the default printer.

Library

UTLFUNC.PBL

Syntax

f_print_file (*printfilename*)

Parameter	Description
printfilename	String containing the fully qualified name of the file to be printed

Return value

Boolean. Returns TRUE if the function succeeded and FALSE if it did not.

Usage

Use this function to print a text file.

It is assumed that the file to be printed is composed of plain ASCII text with normal end-of-line characters (carriage return, line feed).

Example

This example prints the file named in the string instance variable is_filename. If the print fails, then an error message is displayed.

```
string   is_filename

IF not f_print_file (is_filename) THEN
   MessageBox("PRINT ERROR", &
      "An error occured  during the printing of " &
      + is_filename + ".")
END IF
```

f_promptforcriteria

Description

Allows you to enable or disable the Prompt for Criteria dialog box for specified columns

Library

UTLFUNC.PBL

Syntax

f_promptforcriteria (*dwcontrol, columnnames, yesno*)

Parameter	Description
dwcontrol	DataWindow variable of the DataWindow control containing the columns to be returned
columnnames	Unbounded string array containing the names of columns affected by the function
yesno	String specifying either yes (enable prompt for criteria) or no (disable prompt for criteria)

Return value

Integer. Returns *1* if the function succeeds and *-1* if it does not.

Usage

Use this function to control display of the Prompt for Criteria dialog box.

Example

This example enables display of the Prompt for Criteria dialog box for the emp_id column.

```
integer  li_return
string   ls_cols[]

ls_cols[1] = "emp_id"

li_return = &
   f_promptforcriteria ( dw_sheet, ls_cols, "YES" )
```

f_push_parm

Description

Adds a string value to the parameter stack that is contained in the window w_hold_parms.

The main purpose of this window (to pass parameters when opening and closing windows) has been replaced with the functionality provided by OpenWithParm and CloseWithReturn functions. Global function f_push_parm has been retained for backward compatibility.

Library

UTLFUNC.PBL

Syntax

f_push_parm (*newstackvalue*)

Parameter	Description
newstackvalue	String containing the value to be added to the parameter stack

Return value

None

Usage

This function is provided for backward compatibility. New applications should use the OpenWithParm and CloseWithReturn functions instead.

> **Note**
> To use this function, you must open the w_hold_parms window in the application's Open event.

Example

This example uses f_push_parm to pass title and the error message to an error handling routine from the window that had the error.

```
string ls_title, ls_message

ls_title = " OPEN WINDOW EVENT "
ls_message = "Error in the Open window event."
f_push_parm (ls_title)
f_push_parm (ls_message)
```

See also

f_get_parm
f_pop_parm
f_set_parm
w_hold_parms

f_referential_int

Description

Allows you to determine referential integrity based on a passed SQL statement. Use it for databases that do not support referential integrity.

Library

UTLWIN.PBL

Syntax

f_referential_int (*sqlstatement, inputparm*)

Parameter	Description
sqlstatement	String containing the SQL statement used to determine the referential integrity
inputparm	String containing the variable value used by the SQL statement

Return value

Long. Returns the number of rows retrieved and *-1* if there was an error executing the statement.

Usage

Call this function before allowing the user to delete a row from a DataWindow. For example, you might use this function to ensure that no employees exist in a department before deleting the department.

Example

This example determines if any records exist in the EMPLOYEE table where the DEPT_ID is equal to the passed value. If an error is found with the SQL, the function returns *-1*; otherwise, it returns the number of employee rows found. If the number of rows is greater than 0, there is a referential integrity problem and the delete is disallowed.

```
string   ls_sql_statement, ls_value
long     ll_rows

ls_sql_statement = &
   "Select * from EMPLOYEE where DEPT_ID = ?"
ls_value = "400"
ll_rows=f_referential_int(ls_sql_statement,
ls_value)
CHOOSE CASE ll_rows
CASE -1
   f_error_box("SQL Error", &
     "SQL Error in f_referential_integrity
function")
CASE 0
   DeleteRow()
CASE >0
   f_error_box("Integrity Error", &
      "Referential Integrity Violation: " &
     + "Department still has employees.")
END CHOOSE
```

f_retrieve_dddw

Description

Retrieves rows for a specified dropdown DataWindow column.

Library

UTLFUNC.PBL

Syntax

f_retrieve_dddw (*dwcontrol, columnname, transactionobject*)

Parameter	Description
dwcontrol	DataWindow variable of the DataWindow control that contains the dropdown DataWindow
columnname	String specifying the name of the column that contains the dropdown DataWindow
transactionobject	Transaction variable specifying the transaction object

Return value

Boolean. Returns TRUE if the function succeeds and FALSE if it does not.

Usage

Call this function in a window's Open event to retrieve rows for a dropdown DataWindow.

Example

This example retrieves data for the dropdown DataWindow that contains the state column.

```
IF NOT f_retrieve_dddw(dw_sheet,"state",SQLCA) THEN
   MessageBox("Retrieve","Retrieval error: state")
END IF
```

f_right_justify

Description

Returns a string of specified length that is right justified, by adding spaces to the left of the original string. If the string is longer than the requested length, the original string is returned.

Library

UTLFUNC.PBL

Syntax

f_right_justify (*originalstring, length*)

Parameter	Description
originalstring	String to be right-justified
length	Integer representing the length of the string

Return value

String containing the right-justified value.

Usage

Use this function to right-justify a string that contains numbers.

This global function is necessary when printing information because the String function does not provide the capability to format numbers or to right-justify strings.

Example

This example shows how to right-justify the value of the variable li_foo in a 10-character field.

```
string ls_output
integer li_foo

li_foo = 100
ls_output = string(li_foo)
// Right-justify in a 10-character variable.
ls_output = f_right_justify(ls_output,10)
```

f_select_data

Description

Opens the w_dw_select window, which allows a user to select a row from a DataWindow. The information from the selected row is passed back to the calling window. The data displayed by the w_dw_select window is determined by the SQL statement that you pass to the function. The function uses this SQL statement to create a DataWindow dynamically.

Library

UTLWIN.PBL

Syntax

f_select_data (*sql, title, columns, dwcontrol, row, autoquery*)

Parameter	Description
sql	String containing SQL statement used to build a DataWindow.
title	String containing title for w_dw_select window.
columns	String specifying column mapping. Composed of one or more pairs of column names from the calling DataWindow equated to a column number in the passed SQL statement. This mapping defines which columns in the calling DataWindow will be updated.
dwcontrol	DataWindow variable identifying the DataWindow control that is to be updated when the user selects a row in the selection window.
row	Long identifying the DataWindow row that is to be updated when the user selects a row in the selection window.
autoquery	Boolean indicating whether to automatically retrieve the data in the selection window (TRUE) or allow the user to enter retrieval criteria (FALSE).

Return value

None

Usage

You typically use this global function to allow users to double-click in a DataWindow and display a popup selection list from which they can select rows to be returned to the calling DataWindow.

Example

Use the following code in the DoubleClicked event of a DataWindow. It displays a select list in the w_dw_select window based on the contents of the title table, allows the user to enter retrieval arguments (query_mode), and then selects one of the displayed rows. Then it moves the data from the selected row back into the calling DataWindow at the row that was double-clicked. Columns title_id, title, price, ytd_sales, and type are filled with new values.

```
string col_name
long row

col_name = GetObjectAtPointer()
col_name = f_get_token(col_name,"~t")

IF col_name = 'title_id' THEN
   row = GetClickedRow()
   f_select_data ("SELECT title_id,title, price,&
      ytd_sales, type FROM titles",  &
```

```
        "Select Title", "title_id=1, title=2,
price=3,&
        ytd_sales=4, type=5", this,row, false )
END IF
```

See also w_dw_select

f_set_menu_branch

Description Sets the attributes of all items under a give menu branch to a given state.

Library UTLFUNC.PBL

Syntax **f_set_menu_branch** (*menubranch, operation*)

Parameter	Description
menubranch	Menu variable of the branch whose menu items will be affected
operation	String specifying the operation to perform on all menu items under *menubranch*: ♦ Check ♦ Uncheck ♦ Enabled ♦ Disabled ♦ Visible ♦ Invisible

Return value Integer indicating the number of menu items that were modified.

Usage Call this function to set all menu items under a menu parent to the same state.

Example This example calls the f_set_menu_branch function and set all items under the Actions menu be disabled.

```
f_set_menu_branch(m_share.m_topicactions,"disabled")
```

f_set_parm

Description

Adds or updates the w_hold_parms DataWindow using the value of the passed parameter.

The main purpose of this window (to pass parameters when opening and closing windows) has been replaced with the functionality provided by OpenWithParm and CloseWithReturn functions. Global function f_set_parm has been retained for backward compatibility.

Library

UTLFUNC.PBL

Syntax

f_set_parm (*itemindicator, itemvalue*)

Parameter	Description
itemindicator	String representing the item in the stack that will be added or updated
itemvalue	String containing a value for *itemindicator*

Return value

None

Usage

This function is provided for backward compatibility. New applications should use the OpenWithParm and CloseWithReturn functions instead.

> **Note**
> You must open the w_hold_parms window in the application's Open event.

Example

This example sets the item PRODUCT_NAME to Jet Fuel in the parameter stack.

```
f_set_parm(PRODUCT_NAME,"Jet Fuel")
```

See also

f_get_parm
f_pop_parm
f_push_parm
w_hold_parms

f_set_sqlca

Description

Opens the window w_set_sqlca, which allows the user to enter SQLCA-specific DBMS information.

Library

UTLWIN.PBL

Syntax

f_set_sqlca ( )

Return value

None

Usage

This function is typically called in the Open event of the application object when there is no initialization information available to the application. Use it as an alternative method of establishing database connection information.

This function is best used in the development and testing environment. Other methods (such as establishing an application INI file) are preferred in the production environment since they do not require the user to enter database-specific information.

Example

This example calls the f_set_sqlca global function.

```
f_set_sqlca()
```

See also

w_set_sqlca

f_sort_order

Description

Displays the passed DataWindow in the w_sort_order window, which allows the user to control DataWindow sort order.

Library

UTLWIN.PBL

Syntax

f_sort_order (*datawindowname, sortcolumns*)

Parameter	Description
datawindowname	DataWindow variable indicating the DataWindow for which the sort order is specified.
sortcolumns	String containing the possible sort columns on the DataWindow that a user can choose. Use the format **heading:column name,heading:column name, heading:column name.** You can pass up to 10 columns that the user may specify to sort. If the DataWindow is already sorted, the sort window will display the current sort order.

Return value

Boolean. Returns TRUE if it succeeds and FALSE if it fails.

Usage

Use this function as a way of allowing users to control DataWindow sort order.

> **DataWindow should not use grouping**
> Unpredictable results can occur if the DataWindow to be sorted contains grouping.

Example

This example opens the window w_sort_order, which allows the user to choose from the columns in dw_1 (employee name, department, and employee number) to sort in either ascending or descending order. The user can choose more than one sort column.

```
f_sort_order(dw_1, "Employee Name:emp_name," &
   + "Department:dept_named,Employee
Number:emp_num")
```

See also

w_sort_order

f_string_to_boolean

Description

Returns a boolean value denoting TRUE or FALSE based on the specified string.

Library UTLFUNC.PBL

Syntax **f_string_to_boolean** (*evaluationvalue*)

Parameter	Description
evaluationvalue	String containing the value to evaluate. This value can be in either uppercase or lowercase.

Return value Boolean. Returns TRUE if *evaluationvalue* starts with T or Y and FALSE if it does not.

Usage Use this function to convert strings into boolean values.

Example This example uses the f_string_to_boolean function to convert the visible attribute to a boolean data type. The two f_get_token functions strip off the beginning of the results variable, leaving the visible attribute remaining. This contains a string with either "TRUE" or "FALSE" and f_string_to_boolean sets the lb_visible variable to the appropriate boolean value.

```
string results
integer li_x, li_y
boolean lb_visible

results = &
   f_dw_get_attributes(dw_1,'au_id','x,y,visible')

li_x = integer(f_get_token(results,'~n'))
li_y = integer(f_get_token(results,'~n'))
lb_visible = f_string_to_boolean(results)
```

See also f_boolean_to_string
f_dw_get_attributes
f_get_token

f_time_diff

Description Calculates the difference in milliseconds between two times and returns the difference.

Library UTLFUNC.PBL

Syntax **f_time_diff** (*starttime*, *endtime*)

Parameter	Description
starttime	Time variable containing a valid time that you want to be the beginning time
endtime	Time variable containing a valid time that you want to be the end time

Return value Ulong containing the difference in milliseconds between the two times.

Usage You might find this function useful in performance profiling.

Example This statement sets the value of lu_diff to five milliseconds.

```
time  lt_start, lt_end
ulong lu_diff

lt_start = 10:00:00
lt_end = 10:00:05
lu_diff = f_time_diff(lt_start,lt_end)
```

f_wait_for

Description Opens the response window w_wait_for, which checks every three seconds to determine if a file has appeared or disappeared. This function causes the PowerBuilder script to wait until the application has completed.

Library UTLWIN.PBL

Syntax **f_wait_for** (*filename*, *appeardisappear*)

Parameter	Description
filename	String containing the name of the file. If *filename* is not on the current application library search path, enter the fully qualified name.

Parameter	Description
appeardisappear	Boolean variable indicating whether to wait for file to appear or to disappear: ♦ TRUE — Wait for the named file to appear ♦ FALSE — Wait for the named file to disappear

Return value

Boolean. TRUE indicates that the file appeared or disappeared as specified in *appeardisappear.* and FALSE indicates that the user clicked the Cancel button on the w_wait_for window.

Usage

Use this function to serialize the running of external applications or when your application depends on the presence or absence of a particular file.

Example

This example runs a process named work.pif, which writes a file named done.xxx when it is completed. The script will resume running when this file appears.

```
boolean status

// Make sure that the semaphore file does not exist.
FileDelete('done.xxx')
Run("work.pif")
// Wait for the file 'done.xxx' to appear.
status = f_wait_for('done.xxx',TRUE)
IF NOT status THEN
   // Error on DOS process or user canceled.
END IF
```

See also

w_wait_for

f_write_file

Description

Writes the passed string to the designated filename.

Library

UTLFUNC.PBL

Syntax

f_write_file (*filename, writestring, writemode*)

Parameter	Description
filename	String containing the fully qualified name of the file to be written
writestring	String that will be written to the filename
writemode	Writemode enumerated data type (Append! or Replace!)

Return value

Boolean. Returns TRUE if the write succeeds and FALSE if it fails.

Usage

Use this function to write to a file.

Example

This statement appends the mle_description.text to the file SPECS.TXT.

```
IF NOT &
   f_write_file("SPECS.TXT",mle_1.text,Append!) THEN
   Message box("Error ","Could not append to file")
END IF
```

See also

f_write_log

f_write_log

Description

Appends the passed string to the end of the passed file name. It adds a carriage control/line feed to the end of the passed string.

Library

UTLFUNC.PBL

Syntax

f_write_log (*filename, writetext*)

Parameter	Description
filename	String containing the fully qualified name of the file that will have the text appended to it
writetext	String containing the text to be appended to *filename*

Return value

Boolean. Returns TRUE if the write succeeds and FALSE if it fails.

Usage

Use this function to append text to a log file.

Example

This example appends a text string to the file namedate.log. The string contains the name of the person and date they logged in to an application. The name and date are values passed from the log in screen.

```
string ls_person_date, ls_person_name

ls_person_name = "Martha DeMers "
ls_person_date = ls_person_name + String( Today() )
f_write_log ("c:\appl\namedate.log",ls_person_date)
```

See also

f_write_file

CHAPTER 7

Global Structures

About this chapter

This chapter describes the global structures in the Application Library.

str_frame

Description

Contains application information.

Field name	Data type
app_name	Application
ini_file_name	String
allow_multiple_apps	Boolean

Library

SYS.PBL

Usage

This structure is used by scripts in the w_sys_frame window. You do not typically use it in your applications.

See also

w_select
w_sys_mast_detl_dw
w_sys_single_dw

str_parms

Description

Contains all-in-one structure arrays of simple data types.

Field name	Data type
string_arg[ ]	String
char_arg[ ]	Char
int_arg[ ]	Integer
long_arg[ ]	Long
date_arg[ ]	Date
datetime_arg[ ]	Datetime
time_arg[ ]	Time
boolean_arg[ ]	Boolean
real_arg[ ]	Real
decimal_arg[ ]	Decimal {2}
canceled	Boolean
double_arg[ ]	Double

Library

UTLWIN.PBL

Usage

You can use this structure in conjunction with the OpenWithParm and OpenSheetWithParm functions to pass parameters between windows.

See also

w_select
w_sys_mast_detl_dw
w_sys_single_dw

str_progress

Description

Passes parameters to window object w_progress. It contains the following data types:

Field name	Data type
cancel_window	Window
cancel_event	String
title	String

Library

UTLWIN.PBL

Usage

This structure is used by scripts in the w_progress window. You do not typically use it in your applications.

See also

w_progress

str_select_parms

Description

Passes parameters to window object w_dw_select.

Field name	Data type
sql	String
title	String
columns	String
dw	DataWindow
row	Long
auto_query	Boolean

Library

UTLWIN.PBL

Usage

This structure is used by scripts in the w_dw_select window. You do not typically use it in your applications.

See also

w_dw_select

str_sort

Description

Passes parameters to window object w_sort.

Field name	Data type
dw	DataWindow
title	String

Library

UTLWIN.PBL

Usage

This structure is used by scripts in the w_sort window. You do not typically use it in your applications.

See also

w_sort

str_sort_order

Description

Passes parameters to window object w_sort_order.

Field name	Data type
sort_info	String
dw	DataWindow

Library

UTLWIN.PBL

Usage

This structure is used by scripts in the w_sort_order window. You do not typically use it in your applications.

See also

w_sort_order

CHAPTER 8

Menu Objects

About this chapter

This chapter describes the menu objects in the Application Library.

m_base

Description

Basic ancestor menu that contains a file menu and a window menu. You can define additional menu items in the descendent menus.

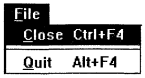
Library

UTLWIN.PBL

Usage

Use the m_base menu object as a basic ancestor menu.

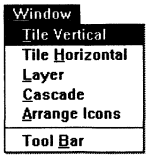
File menu



The File menu has predefined menu items that you can use to control the window.

Menu item	Description
Close	Closes the current window.
	Shortcut key: CTRL+F4
Quit	Exits the application.
	Shortcut key: ALT+F4

Window menu



The Window menu is a standard for MDI applications.

Menu item	Description
Cascade	Uses the ArrangeSheets function to cascade the sheets that are not minimized
Tile Horizontal	Uses the ArrangeSheets function to tile the sheets that are not minimized from side-to-side
Tile Vertical	Uses the ArrangeSheets function to tile the sheets that are not minimized one above the other
Layer	Uses the ArrangeSheets function to layer the sheets that are not minimized
Arrange Icons	Uses the ArrangeSheets function to arrange the icons for minimized sheets
Tool Bar	Opens the w_set_toolbars window, which allows the user to control toolbar display

m_sys_frame

Description

Ancestor menu for applications that use the application framework. It contains File, Application Topics, Actions, Window, and Help menus.

Many m_sys_frame menu items trigger user events in application framework windows. The individual discussions note any menu items that are particular to a single application framework window.

Library

SYS.PBL

Menu functions

mf_frame Returns the frame window.

- *Parameters.* None
- *Return value.* Window.

mf_set_menu_item Sets the specified menu item to the specified state and shows or hides the corresponding toolbar button.

- *Parameters*:
  - Menu variable specifying the menu item to be enabled or disabled.
  - Boolean specifying whether to enable (TRUE) or disable (FALSE) the menu item.
- *Return Value*. None.

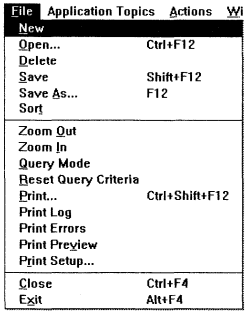
mf_set_query_mode Enables or disables query mode.

- *Parameters*. Boolean specifying whether to enable (TRUE) or disable (FALSE) query mode.
- Return Value. None.

Usage

Use the m_sys_frame menu object as the top-level ancestor menu for all windows created using the application framework..

File menu



The File menu has predefined menu items that you can use for window, file, and database actions. Although this section documents all menu items for the m_sys_frame File menu, you typically enable or disable items in this menu to match the unique needs of each descendent menu.

Menu item	Description	Toolbar button
New	Adds a new row in the dw_sheet DataWindow by triggering the window's ue_filenew user event.	
Open	Allows the user to select a new row to display by triggering the window's ue_fileopen user event (ue_fileopen triggers the ue_select_window user event, to which you can add selection logic). Shortcut key: CTRL+F12	

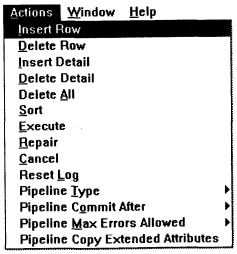
Menu item	Description	Toolbar button
Delete	Deletes the current row in the dw_sheet DataWindow by triggering the window's ue_filedelete user event. This deletes the row from the DataWindow and in some cases deletes the row from the database. See the ancestor window's ue_filedelete user event for more information.	
Save	Saves all new and modified rows to the database by triggering the window's ue_filesave user event. Shortcut key: SHIFT+F12	
Save As	Triggers the window's ue_filesaveas user event, which uses the SaveAs function to save the DataWindow contents to a file. Shortcut key: F12	
Sort	Triggers the window's ue_sort user event, which opens the w_sort window to allow the user to specify a sort order for the multirow DataWindow.	
Zoom Out	Triggers the window's ue_zoom_smaller user event, which shrinks the DataWindow. This menu item is designed to work with descendants of w_sys_report.	
Zoom In	Triggers the window's ue_zoom_bigger user event, which enlarges the DataWindow. This menu item is designed to work with descendants of w_sys_report.	
Query mode	Toggles the DataWindow into or out of query mode by triggering the window's ue_toggle_query_mode user event. This menu item is designed to work with descendants of w_sys_report. The toolbar button changes from a hand to a report in query mode.	 
Reset Query Criteria	Resets query criteria by triggering the window's ue_reset_query_criteria user event. This menu item is designed to work with descendants of w_sys_report.	

Menu item	Description	Toolbar button
Print	Prints the dw_sheet DataWindow by triggering the window's ue_fileprint user event. Shortcut key: CTRL+SHIFT+F12	
Print Log	Prints the dw_msg DataWindow by triggering the window's ue_print_log user event. Use this menu item with descendants of w_sys_pipeline.	
Print Errors	Prints the dw_pipe_errors DataWindow by triggering the window's ue_print_errors user event. Use this menu item with descendants of w_sys_pipeline.	
Print Preview	Triggers the window's ue_fileprint_preview user event, which opens the w_printzoom window. This menu item is designed to work with descendants of w_sys_report.	
Print Setup	Uses the PrintSetup function to display the Printer Setup dialog box.	
Close	Closes the current window by triggering the window's ue_fileclose user event. Shortcut key: CTRL+F4	
Exit	Exits the application by closing the frame window. Shortcut key: ALT+F4	

Application Topics menu

The Application Topics menu has no menu items defined. You can add items at the descendent level when creating a new menu that is inherited from m_sys_frame. These items are based on the application requirements and are normally used to allow the opening of application-related sheets.

Actions menu



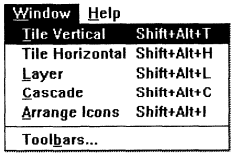
The Actions menu at the ancestor level contained predefined menu items for use with application framework descendent windows. You can add items at the descendent level when creating a new menu that is inherited from m_sys_frame. These items are defined based on the application requirements. Actions are typically dependent on the active sheet. Items added might include opening a sheet related to the active sheet or other specific processing.

Menu item	Description	Toolbar button
Insert Row	Adds a new row in the dw_sheet DataWindow by triggering the window's ue_filenew user event.	
Delete Row	Deletes the current row in the dw_sheet DataWindow by triggering the window's ue_filedelete user event. This deletes the row from the DataWindow and in some cases deletes the row from the database.	
Insert Detail	Adds a new row in the dw_detail DataWindow by triggering the window's ue_detail_new user event. Use this menu item with descendants of the w_sys_mast_detl_dw window.	
Delete Detail	Deletes the current row in the dw_detail DataWindow by triggering the window's ue_delete_detail user event. Use this menu item with descendants of the w_sys_mast_detl_dwwindow.	
Delete All	Deletes all rows in the dw_sheet DataWindow by triggering the window's ue_deleteall user event. The descendent window is responsible for maintaining data integrity. Use this menu item with descendants of the w_sys_multi_dw and w_sys_shared_dw windows.	
Sort	Triggers the window's ue_sort user event, which opens the w_sort window to allow the user to specify a sort order for the multirow DataWindow.	

Menu item	Description	Toolbar button
Execute	Triggers the window's ue_execute_pipe user event, which executes a pipeline. Use this menu item with descendants of the w_sys_pipeline window.	
Repair	Triggers the window's ue_repair_pipe user event, which submits corrected rows to the destination database. Use this menu item with descendants of the w_sys_pipeline window.	
Cancel	Triggers the window's ue_cancel_pipe user event, which halts pipeline execution. Use this menu item with descendants of the w_sys_pipeline window.	
Reset Log	Triggers the window's ue_reset_log user event, which clears rows from the dw_msg DataWindow. Use this menu item with descendants of the w_sys_pipeline window.	
Pipeline Type	Displays a cascading menu with a list of pipeline types, from which you can choose one. Pipeline types are Create, Replace, Refresh, Append, and Update. Selecting a pipeline type triggers the window's ue_set_pipe_type user event. Use this menu item with descendants of the w_sys_pipeline window.	
Pipeline Commit After	Displays a cascading menu with commit frequencies, from which you can choose one. COMMIT frequencies are All, 1, 10, 100, and 1,000. Selecting a COMMIT frequency triggers the window's ue_set_pipe_commit user event. Use this menu item with descendants of the w_sys_pipeline window.	
Pipeline Max Errors Allowed	Displays a cascading menu with a list of pipeline types, from which you can choose one. Max errors values are None, 1, 10, 100, and 1,000. Selecting a max errors value triggers the window's ue_set_pipe_maxerrors user event. Use this menu item with descendants of the w_sys_pipeline window.	

Menu item	Description	Toolbar button
Pipeline Copy Extended Attributes	Enables or disables the copying of database extended attributes. Selecting this option triggers the window's ue_set_pipe_ext_attr user event. Use this menu item with descendants of the w_sys_pipeline window.	

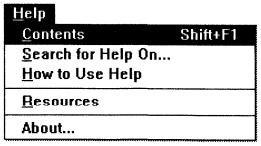
Window menu



The Window menu is standard for an MDI application. Each menu item has a script to perform the required processing. These scripts are inherited by a descendent menu. The Toolbars item allows the user to manipulate the toolbar.

Menu item	Description
Tile Vertical	Uses the ArrangeSheets function to tile the sheets that are not minimized one above the other. Shortcut key: SHIFT+ALT+T
Tile Horizontal	Uses the ArrangeSheets function to tile the sheets that are not minimized from side-to-side. Shortcut key: SHIFT+ALT+H
Layer	Uses the ArrangeSheets function to layer the sheets that are not minimized. Shortcut key: SHIFT+ALT+L
Cascade	Uses the ArrangeSheets function to cascade the sheets that are not minimized. Shortcut key: SHIFT+ALT+C
Arrange Icons	Uses the ArrangeSheets function to arrange the icons for minimized sheets. Shortcut key: SHIFT+ALT+I
Toolbars	Opens the w_set_toolbars window, which allows the user to control toolbar display.

Help menu



The Help menu is a standard menu item and is located at the end of the menu bar. You must add script (typically using the ShowHelp function) to access your application-specific Windows help file.

For an example of enabling the first three menu items, see Lesson 5, "Building a Menu for the Frame Window" in Part Two.

Menu item	Description	Toolbar button
Contents	Typically used to display an application-specific Windows Help file.	
Search for Help On	Typically used to display the Search dialog box for an application-specific Windows Help file.	
How to Use Help	Typically used to display the WINHELP.HLP file. Use the following function as an example: `ShowHelp("winhelp.hlp",Index!)`	
Resources	Opens the w_get_free_resources_graph window, which display free system resources.	
About	Displays the w_about window. You can override this at the descendent level to display an application-specific About window.	

CHAPTER 9

User Objects

About this chapter

This chapter describes the user objects in the Application Library.

u_help_bar

Description

Simulates a MicroHelp bar on a non-MDI frame window. You can place this object on any non-MDI frame window.

Library

UTLWIN.PBL

Instance variables

Variable	Data type	Access
iw_parent_window	Window	Public
ii_menu_ht	Integer	Public
ib_show_clock	Boolean	Public
ii_resizeable_offset	Integer	Public

User object functions

The user object functions encapsulated in u_help_bar are:

- Uf_init, explained on page 285.
- Uf_resized, explained on page 285.
- Uf_set_clock, explained on page 286.
- Uf_set_msg, explained on page 286.

Usage

You can use the u_help_bar user object to provide information to a user, much like using the SetMicroHelp function on an MDI frame window. To use the u_help_bar user object:

1 Add the user object control to the bottom of your non-MDI window and optionally rename it (uo_help_bar is the recommended name).

2 In the window's Open event, call the uf_init, uf_set_msg, and uf_set_clock (optional) user object functions. If you are displaying the clock, also code a Timer function (60 seconds is the recommended interval).

3 In the window's Resize event, call the uf_resized user object function.

4 If you are displaying the clock, call the uf_set_clock user object function in the window's Timer event.

5 Modify the u_help_bar message as necessary by calling uf_set_msg.

Examples

In the window's Open event, add these statements:

```
uo_help_bar.uf_init(this,TRUE)
uo_help_bar.uf_set_clock()
uo_help_bar.uf_set_msg("Your message here")
Timer(60,this)
```

In the window's Resize event, add this statement:

```
uo_help_bar.uf_resized()
```

In the window's Timer event, add this statement:

```
uo_help_bar.uf_set_clock()
```

> **Use the control name, not the object name**
> In the parent window scripts, you use dot notation to qualify u_help_bar user object functions with the *control* name (for example, uo_help_bar), not the object name (u_help_bar).

See also

w_mdi_clock

uf_init

Description

Registers the window with the user object. It also determines its size and placement, optionally displaying the date and time if the *displaydatetime* boolean variable is TRUE.

Syntax

controlname.**uf_init** (*windowname*, *displaydatetime*)

Parameter	Description
controlname	The window control name of the u_help_bar user object.
windowname	Window variable indicating the window in which to place u_help_bar.
displaydatetime	Boolean. TRUE indicates that the date and time display on the right hand side of the bar, FALSE indicates that data and time do not display.

Return value

None

Usage

Call this user object function in the Open event of the parent window. You only need to call it once.

Example

This example calls the function uf_init and passes the window name that the object is placed on and a boolean variable equal to TRUE.

```
uo_help_bar.uf_init(this, TRUE)
```

uf_resized

Description

Places the user object on a resized window.

Syntax

controlname.**uf_resized** ( )

Parameter	Description
controlname	The window control name of the u_help_bar user object

Return value

None

Usage

Call this user object function in the Resize event of the parent window.

Example

This statement calls the uf_resized function, which positions the user object on the resized window.

```
uo_help_bar.uf_resized( )
```

uf_set_clock

Description

Refreshes the clock information with the current date and time.

Syntax

controlname.**uf_set_clock** ( )

Parameter	Description
controlname	The window control name of the u_help_bar user object

Return value

None

Usage

Call this user object function in the Timer event of the parent window.

Example

This example calls the uf_set_clock function.

```
// Refresh the displayed time and date.
uo_help_bar.uf_set_clock()
```

uf_set_msg

Description

Sets a display message on the left side of the u_help_bar user object.

Syntax

controlname.**uf_set_msg** (*message*)

Parameter	Description
controlname	The window control name of the u_help_bar user object.
message	String containing the text to be displayed on the left side of the bar. To clear the display, specify an empty string ("").

Return value None

Usage Call this user object function from any event script in the parent window.

Example This example calls the function uf_set_msg and sets the u_help_bar display message.

```
string   ls_msg

ls_msg = "Updates have been applied."
uo_help_bar.uf_set_msg(ls_msg)
```

u_mdi_clock_item

Description Contains text used as a cell in the w_mdi_clock window..

Library UTLWIN.PBL

User object functions The user object functions encapsulated in u_mdi_clock_item are:

- Uf_set_text, explained on page 288.
- Uf_set_width, explained on page 288.

Usage The u_mdi_clock_item is used by the w_mdi_clock window to contain the blocks of text displayed on the right. Access u_mdi_clock_item and its user object functions through w_mdi_clock window functions.

Example For examples of using u_mdi_clock_item, see the wf_add_item, wf_del_item, and wf_set_text window functions in the w_mdi_clock window.

See also w_mdi_clock

uf_set_text

Description

Sets the text contained in u_mdi_clock_item.

Syntax

controlname.**uf_set_text** (*text*)

Parameter	Description
text	String variable containing the text to be displayed in u_mdi_clock_item

Return value

None

Usage

Call this user object function to specify text to display in u_mdi_clock_item.

Example

For an example of uf_set_text, see the w_mdi_clock's wf_set_text window function.

uf_set_width

Description

Specifies the width of u_mdi_clock_item.

Syntax

controlname.**uf_width** (*width, alignment*)

Parameter	Description
width	Integer specifying a width for u_mdi_clock_item
alignment	Alignment enumerated variable type specifying alignment for u_mdi_clock_item text (center!, left!, or right!)

Return value

None

Usage

Call this user object function to specify width and alignment for u_mdi_clock_item.

Example

For an example of uf_set_width, see the w_mdi_clock's wf_add_item window function.

u_ole

Description

Standard visual user object of type OLEControl. This user object provides basic functions for OLE usage. Place this user object in a window instead of an OLE 2.0 control.

U_ole is the ancestor object for the u_ole_excel and u_ole_word user objects, explained later in this chapter.

For information on using OLE, see *Building Applications* in the PowerBuilder documentation set. For information on OLE attributes, events, and functions, see *Objects and Controls* in the PowerBuilder documentation set.

Library

SYS.PBL

User object functions

The user object functions encapsulated in u_ole are:

- Uf_load, explained on page 289.
- Uf_save, explained on page 290.
- Uf_saveas, explained on page 291.

Usage

You use the u_ole user object to load an object into an OLE control and to save an object as a binary large object (BLOB).

uf_load

Description

Loads a file into an OLE control. This file can be a server file with an extension that is related to an OLE server application (for example, XLS is related to Microsoft Excel) or a binary large object (BLOB) with the OLE extension.

Syntax

olecontrol.**uf_load** (*filename*)

Parameter	Description
olecontrol	Name of the window control containing the u_ole user object
filename	String naming the file to be loaded

Return value

None.

Usage

Call this user object function to load a file into an OLE control. Users can then activate the server application using the method associated with the control (usually double-clicking).

Example

This example calls the uf_load function to load the Q1SALES.XLS file into the OLE control.

```
ole_1.uf_load("Q1SALES.XLS")
```

> **Use the control name, not the object name**
> In the parent window scripts, you use dot notation to qualify u_ole user object functions with the *control* name (for example, ole_1), not the object name (u_ole).

uf_save

Description

Saves the contents of u_ole as a BLOB (OLE file extension).

Syntax

olecontrol.**uf_save** ( )

Parameter	Description
olecontrol	Name of the window control containing the u_ole user object

Return value

None.

Usage

Call this user object function to save the contents of u_ole as a BLOB (OLE extension). To use this function, you must have already loaded the control's contents from an OLE file. If you want to save a server file (XLS or DOC file) as an OLE file, use the uf_saveas function.

Example

This example calls the uf_save function.

```
ole_1.uf_save( )
```

uf_saveas

Description

Saves the contents of u_ole as a BLOB (OLE file extension).

Syntax

olecontrol.**uf_saveas** (*filename*)

Parameter	Description
olecontrol	Name of the window control containing the u_ole user object.
filename	String containing the name of the file to be saved. If you do not provide a filename, the function will prompt for one.

Return value

None.

Usage

Call this user object function to save the contents of u_ole as a BLOB.

Example

This example calls the uf_saveas function.

```
ole_1.uf_saveas("Q1SALES.OLE")
```

u_ole_excel

Description

Standard visual user object of type OLEControl. This user object, which is inherited from u_ole, provides basic functions for accessing Microsoft Excel. Place this user object in a window instead of an OLE 2.0 control.

Library

SYS.PBL

User object functions

The user object functions encapsulated in u_ole_excel are:

- Uf_getvalue, explained on page 292.
- Uf_setfocus, explained on page 293.
- Uf_setvalue, explained on page 294.

Usage

You use the u_ole_excel user object to access Microsoft Excel spreadsheet data through a PowerBuilder window control.

uf_getvalue

Description

Returns the value in the specified spreadsheet cell.

Syntax

olecontrol.**uf_getvalue** (*row, column*)

Parameter	Description
olecontrol	Name of the window control containing the u_ole_excel user object
row	Integer specifying the row location
column	Integer specifying the column location

Return value

String containing the value in the specified cell.

Usage

Call this user object function to access spreadsheet values.

Example

This example calls the uf_getvalue function to return the value at spreadsheet cell 4,4.

```
string   ls_cell_value

ole_1.uf_load("Q1SALES.XLS")
ls_cell_value = ole_1.uf_getvalue(4,4)
```

uf_setfocus

Description

Sets focus to the specified spreadsheet cell.

Syntax

olecontrol.**uf_setfocus** (*row, column*)

Parameter	Description
olecontrol	Name of the window control containing the u_ole_excel user object
row	Integer specifying the row location
column	Integer specifying the column location

Return value

Boolean. Returns TRUE if the function succeeded and FALSE if it did not.

Usage

Call this user object function to set focus to an individual spreadsheet cell.

Example

This example calls the uf_setfocus function to set focus to cell location 4,4.

```
ole_1.uf_load("Q1SALES.XLS")
IF NOT ole_1.uf_setfocus(4,4) THEN
   MessageBox("Setfocus Failed","Can't set focus")
END IF
```

uf_setvalue

Description

Sets a value in the current spreadsheet cell.

Syntax

olecontrol.**uf_setvalue** (*value*)

Parameter	Description
olecontrol	Name of the window control containing the u_ole_excel user object.
value	Value of data type Any that contains the value to be set. You must ensure that the data type passed matches the data type for the current cell.

Return value

None.

Usage

Call this user object function to specify a value for an individual spreadsheet cell.

Example

This example calls the uf_setvalue function to specify a value of 10 to cell location 4,4.

```
any   li_cell_value

ole_1.uf_load("Q1SALES.XLS")
IF NOT ole_1.uf_setfocus(4,4) THEN
   MessageBox("Setfocus Failed","Can't set focus")
END IF
li_cell_value = 10
ole_1.uf_setvalue(li_cell_value)
```

u_ole_word

Description

Standard visual user object of type OLEControl. This user object, which is inherited from u_ole, provides basic functions for accessing Microsoft Word for Windows. Place this user object in a window instead of an OLE 2.0 control.

Library

SYS.PBL

User object functions

The user object functions encapsulated in u_ole_word are:

- Uf_get_bookmarks, explained on page 295.
- Uf_getvalue, explained on page 296.
- Uf_is_bookmark_valid, explained on page 297.
- Uf_setfocus, explained on page 297.
- Uf_setvalue, explained on page 298.

Usage

You use the u_ole_word user object to access Microsoft Word for Windows documents through a PowerBuilder window control.

uf_get_bookmarks

Description

Returns the number of bookmarks in a document and fills an array with bookmark names.

Syntax

olecontrol.**uf_get_bookmarks** (*bookmarklist*)

Parameter	Description
olecontrol	Name of the window control containing the u_ole_word user object.
bookmarklist	Unbounded string array into which the uf_get_bookmarks function places the bookmark names. This array is passed by reference.

Return value

Integer specifying the number of bookmarks in the document (and the number of elements in the *bookmarklist* array).

Usage

Call this user object function to determine the number of bookmarks in a Word document and to determine bookmark names.

Example

This example calls the uf_get_bookmarks function. The li_num_bookmarks variable will contain the number of bookmarks in the current document and the ls_bookmark_names string array will contain bookmark names.

```
string   ls_bookmark_names[ ]
integer  li_num_bookmarks

li_num_bookmarks = &
   ole_1.uf_get_bookmarks(ls_bookmark_names)
```

uf_getvalue

Description

Returns the value in the specified bookmark.

Syntax

olecontrol.**uf_getvalue** (*bookmark*)

Parameter	Description
olecontrol	Name of the window control containing the u_ole_word user object
bookmark	String specifying the bookmark from which to obtain the value

Return value

String containing the value in the specified bookmark.

Usage

Call this user object function to access data from a Microsoft Word for Windows document.

Example

This example calls the uf_getvalue function to return the value at the address1 bookmark.

```
string   ls_bookmark = "address1"
string   ls_bookmark_value

ole_1.uf_load("DISTLETT.DOC")
ls_bookmark_value = ole_1.uf_getvalue(ls_bookmark)
```

uf_is_bookmark_valid

Description

Indicates whether the passed bookmark name exists in the current document.

Syntax

olecontrol.**uf_is_bookmark_valid** (*bookmark*)

Parameter	Description
olecontrol	Name of the window control containing the u_ole_word user object
bookmark	String containing the bookmark name to be tested

Return value

Boolean. Returns TRUE if the string is a bookmark in the current document and FALSE if it is not.

Usage

Call this user object function to determine if a string is a valid bookmark in the current document.

Example

This example calls the uf_is_bookmark_valid function to determine if address1 is a valid bookmark in the current document.

```
string   ls_name = "Jill Jefferson"

IF NOT ole1.uf_is_bookmark_valid("address1") THEN
   MessageBox("Address1","Not a bookmark")
ELSE
   ole_1.uf_setfocus("address1")
   ole_1.uf_setvalue(ls_name)
END IF
```

uf_setfocus

Description

Sets focus to the specified bookmark location.

Syntax

olecontrol.**uf_setfocus** (*bookmark*)

Parameter	Description
olecontrol	Name of the window control containing the u_ole_word user object
bookmark	String specifying the bookmark at which to set focus

Return value

Boolean. Returns TRUE if the function succeeded and FALSE if it did not.

Usage

Call this user object function to set focus to text at a bookmark location.

Example

This example calls the uf_setfocus function to set focus to the bodytext bookmark.

```
ole_1.uf_load("Q1SALES.DOC")
IF NOT ole_1.uf_setfocus("bodytext") THEN
   MessageBox("Setfocus Failed","Can't set focus")
END IF
```

uf_setvalue

Description

Sets the text at the current bookmark location.

Syntax

olecontrol.**uf_setvalue** (*value*)

Parameter	Description
olecontrol	Name of the window control containing the u_ole_excel user object.
value	Value of data type Any that contains the value to be set. You must ensure that the data type passed matches the data type for the current bookmark.

Return value

None.

Usage

Call this user object function to specify text at a particular bookmark location.

Example

This example calls the uf_setvalue function to specify a value of "Jill Jefferson" to the name bookmark.

```
integer  ls_name = "Jill Jefferson"

ole_1.uf_load("Q1SALES.DOC")
IF NOT ole_1.uf_setfocus("name") THEN
   MessageBox("Setfocus Failed","Can't set focus")
END IF
ole_1.uf_setvalue(ls_name)
```

u_pipeline_kit

Description

Class object descended from the Pipeline system object. The u_pipeline_kit user object provides functions for data pipeline usage.

For information on using a data pipeline, see *Building Applications* in the PowerBuilder documentation set. For information on pipeline attributes, events, and functions, see *Objects and Controls* in the PowerBuilder documentation set.

Library

SYS.PBL

Instance variables

Variable	Data type	Access
i_src_trans	Transaction	Private
i_dest_trans	Transaction	Private
ib_elapsed_used	Boolean	Private
ib_executing	Boolean	Private
idw_errors	DataWindow	Private
il_lasttime	Long	Private
il_starttime	Long	Private
ir_total_time	Real	Private
ist_elapsed	StaticText	Private

Variable	Data type	Access
ist_errors	StaticText	Private
ist_read	StaticText	Private
ist_written	StaticText	Private

User object functions

The user object functions encapsulated in u_pipeline_kit are:

- Uf_cancel, explained on page 302.
- Uf_execute, explained on page 302.
- Uf_get_commit, explained on page 304.
- Uf_get_elapsed_time, explained on page 304.
- Uf_get_error_msg, explained on page 305.
- Uf_get_extended_attr_copy, explained on page 306.
- Uf_get_maxerrors, explained on page 306.
- Uf_get_syntax_value, explained on page 307.
- Uf_get_type, explained on page 308.
- Uf_init, explained on page 309.
- Uf_init_elapsed_time, explained on page 310.
- Uf_repair, explained on page 311.
- Uf_set_commit, explained on page 312.
- Uf_set_extended_attr_copy, explained on page 313.
- Uf_set_maxerrors, explained on page 313.
- Uf_set_syntax_value, explained on page 314.
- Uf_set_type, explained on page 315.

Usage

You can use the u_pipeline_kit user object to control a pipeline object. The w_sys_pipeline application framework window uses the u_pipeline_kit user object to provide data pipeline functionality.

For more information on the w_sys_pipeline window, see the "w_sys_pipeline" discussion in Chapter 4.

Although it is recommended that you take advantage of u_pipeline_kit functionality by using a descendant of the w_sys_pipeline window, you can use u_pipeline_kit separately. To use the u_pipeline_kit user object apart from the w_sys_pipeline window:

1 Use the Pipeline painter to create and save a pipeline object.

2 Use the Window painter to create and save a window containing a DataWindow control and four static text controls (these will contain static text for rows read, rows written, rows in error, and elapsed time).

   Perform the remaining steps in your application's events and functions.

3 Create transaction objects for the source and destination databases and connect to these databases.

   If the pipeline copies rows within the same database, you still must define two separate transaction objects.

4 Create an instance of the pipeline object defined in step 1.

5 Initialize the pipeline object with the uf_init function.

6 Execute the pipeline with the uf_execute function.

   If there are errors, PowerBuilder displays them in the DataWindow control.

7 Optionally display pipeline information such as pipeline type, commit level, maximum errors, and elapsed time by using the appropriate functions.

8 Allow the user to fix errors or cancel the process by providing options for the uf_repair and uf_cancel functions.

Examples

Review the w_sys_pipeline window's Open, ue_execute, ue_repair, and ue_cancel events for an example of using u_pipeline_kit.

Use the instance name, not the object name
In the parent window scripts, you use dot notation to qualify u_pipeline_kit user object functions with the *instance* name (for example, i_pipe), not the object name (u_pipeline_kit).

See also

w_sys_pipeline

uf_cancel

Description

Stops pipeline execution.

Syntax

pipelineinstance.**uf_cancel** ( )

Parameter	Description
pipelineinstance	Instance name of the pipeline user object

Return value

Integer. Returns *1* if it succeeds and *-1* if an error occurs.

Usage

Call this user object function after issuing uf_execute or uf_repair functions. The pipeline user object instance remains, but you must reissue the uf_init function before executing the pipeline again.

Example

This example calls the uf_cancel function.

```
Long  ll_return

ll_return = i_pipe.uf_cancel()
IF ll_return = -1 THEN
   MessageBox("Pipeline Cancel","Cancel Failed")
END IF
```

See also

uf_execute

uf_execute

Description

Executes the pipeline object, which transfers data from the source to the destination as specified by the SQL query in the pipeline object.

Syntax

pipelineinstance.**uf_execute** (*message*)

Parameter	Description
pipelineinstance	Instance name of the pipeline user object

Return value

Integer. Returns *1* if it succeeds and a negative number if an error occurs. Error values are:

- *-1* Pipe open failed
- *-2* Too many columns
- *-3* Table already exists
- *-4* Table does not exist
- *-5* Missing connection
- *-6* Wrong arguments
- *-7* Column mismatch
- *-8* Fatal SQL error in source
- *-9* Fatal SQL error in destination
- *-10* Maximum number of errors exceeded
- *-12* Bad table syntax
- *-13* Key required but not supplied
- *-15* Pipe already in progress
- *-16* Error in source database
- *-17* Error in destination database
- *-18* Destination database is read-only

Usage

Call this user object function to transfer data from the source to the destination as specified by the SQL query in the pipeline object named in the uf_init function.

Example

This example calls the uf_execute function.

```
int li_return

li_return = i_pipe.uf_execute( )
IF li_return < 0 THEN
   MessageBox("Pipeline Execute Error", &
      i_pipe.uf_get_error_msg( li_rc ),
Exclamation!)
END IF
```

See also

uf_cancel

uf_get_commit

Description

Returns the number of rows issued between COMMITs.

Syntax

pipelineinstance.**uf_get_commit** ( )

Parameter	Description
pipelineinstance	Instance name of the pipeline user object

Return value

Long indicating the number of rows that will be added before the pipeline issues a COMMIT.

Usage

Call this user object function to determine the COMMIT frequency.

Example

This example calls the uf_get_commit function.

```
long  ll_commit
ll_commit = i_pipe.uf_get_commit( )
```

See also

uf_set_commit

uf_get_elapsed_time

Description

Returns the number of seconds used in the last pipeline execution.

Syntax

pipelineinstance.**uf_get_elapsed_time** ( )

Parameter	Description
pipelineinstance	Instance name of the pipeline user object

Return value

Real indicating the number of seconds used in the last pipeline execution.

Usage

Call this user object function to determine how long the last pipeline execution took.

Example

This example calls the uf_get_elapsed_time function.

```
real  lr_elapsed_time
lr_elapsed_time = i_pipe.uf_get_elapsed_time( )
```

See also

uf_set_elapsed_time

uf_get_error_msg

Description

Returns the error message that corresponds to the passed error number.

Syntax

pipelineinstance.**uf_get_error_msg** (*errornumber*)

Parameter	Description
pipelineinstance	Instance name of the pipeline user object
errornumber	Integer specifying the error number returned by the uf_execute function

Return value

String containing the error message that corresponds to the passed error number.

Usage

Call this user object function to access a message to display in an error window.

Example

This example calls the uf_get_error_msg function.

```
integer li_return

li_return = i_pipe.uf_execute( )
IF li_return < 0 THEN
   MessageBox("Pipeline Execute Error", &

i_pipe.uf_get_error_msg(li_return),Exclamation!)
END IF
```

uf_get_extended_attr_copy

Description

Returns a boolean indicating whether extended attributes will be copied.

Syntax

pipelineinstance.**uf_get_extended_attr_copy** ( )

Parameter	Description
pipelineinstance	Instance name of the pipeline user object

Return value

Boolean. Returns TRUE if extended attributes will be copied and FALSE if they will not.

Usage

Call this user object function to determine whether extended attributes will be copied.

Example

This example calls the uf_get_extended_attr_copy function and updates a checkbox with the returned value.

```
IF i_pipe.uf_get_extended_attr_copy( ) THEN
   cbx_extended_attr.checked = TRUE
ELSE
   cbx_extended_attr.checked = FALSE
END IF
```

See also

uf_set_extended_attr_copy

uf_get_maxerrors

Description

Returns the value of the attribute indicating the number of errors the pipeline will allow before canceling execution.

Syntax

pipelineinstance.**uf_get_maxerrors** ( )

Parameter	Description
pipelineinstance	Instance name of the pipeline user object

Return value

Long indicating the maximum errors that the pipeline will allow before canceling execution.

Usage

Call this user object function to determine the maxerrors value.

Example

This example calls the uf_get_maxerrors function.

```
long  ll_maxerrors
ll_maxerrors = i_pipe.uf_get_maxerrors( )
```

See also

uf_set_maxerrors

uf_get_syntax_value

Description

Returns the value in the pipeline object syntax that corresponds to the passed parameter. This is an internal function that is used by other u_pipeline_kit user object functions; do not call this function directly.

Syntax

pipelineinstance.**uf_get_syntax_value** (*pipelineparm*)

Parameter	Description
pipelineinstance	Instance name of the pipeline user object
pipelineparm	String specifying the parameter whose value is to be returned

Return value

String. Returns the parameter value if it exists in the syntax and an empty string if it does not.

Usage

Call this user object function to determine the value that corresponds to a pipeline object parameter. For example, to find out the name of the source database, pass the string "source_connect" and the function will return the data source name of the source database.

> **Export a pipeline object to view sample syntax**
> To see an example of the parameters used in pipeline syntax, use the library painter to export a pipeline object to a text file. You can then examine the file using the PowerBuilder File Editor, the Windows Notepad, or some other ASCII text editor.

Example

This example calls the uf_get_syntax_value function.

```
string   ls_parm_value

ls_parm_value = i_pipe.uf_get_syntax_value("type")
IF ls_parm_value = "replace" THEN
   MessageBox("Replace", &
      "Table will be replaced. OK to Proceed?", &
      Exclamation!,OKCancel!,2)
END IF
```

See also

uf_get_type
uf_set_syntax_value
uf_set_type

uf_get_type

Description

Returns the pipeline processing type (create, replace, append, and so on).

Syntax

pipelineinstance.**uf_get_type** ( )

Parameter	Description
pipelineinstance	Instance name of the pipeline user object

Return value

String containing the processing type.

- Create
- Replace
- Append
- Refresh
- Update

Usage

Call this user object function to determine the processing type.

Example

This example calls the uf_get_type function.

```
string   ls_type

ls_type = i_pipe.uf_get_ type( )
IF ls_type = "replace" THEN
   MessageBox("Replace", &
      "Table will be replaced. OK to Proceed?", &
      Exclamation!,OKCancel!,2)
END IF
```

See also

uf_get_syntax_value
uf_set_syntax_value
uf_set_type

uf_init

Description

Initializes the pipeline user object instance.

Syntax

pipelineinstance.**uf_init** (*pipelineobject, sourcetransaction, desttransaction, dwcontrol, rowsread, rowswritten, rowsinerror*)

Parameter	Description
pipelineinstance	Instance name of the pipeline user object.
pipelineobject	String containing the name of the data pipeline object defined in the Data Pipeline painter
sourcetransaction	Transaction object with which to connect to the source database
desttransaction	Transaction object with which to connect to the destination database (must be different from *sourcetransaction*)
dwcontrol	DataWindow variable specifying the DataWindow control to contain pipeline errors
rowsread	StaticText window control to contain the number of rows read value
rowswritten	StaticText window control to contain the number of rows written value
rowsinerror	StaticText window control to contain the number of rows in error value

Return value

None

Usage

Call this user object function before executing a pipeline. Before calling uf_init, you must create an instance of u_pipeline_kit, connect to the source database, and connect to the destination database.

Example

This example calls the uf_init function (i_pipe, i_src and i_dest are instance variables; dw_pipe_errors, st_read, st_written, and st_errors are window controls).

```
// I_src will be the source transaction.
i_src = Create transaction
i_src.dbms    = "ODBC"
i_src.dbparm = &
   "ConnectString= 'DSN=Pipe1;UID=dba;PWD=sql'"
CONNECT using i_src;

// I_dest will be the source transaction.
i_dest = Create transaction
i_dest.dbms    = "ODBC"
i_dest.dbparm = &
   "ConnectString= 'DSN=Pipe2;UID=dba;PWD=sql'"
CONNECT using i_dest;

// Create and initialize the pipeline.
i_pipe = Create u_pipeline_kit
i_pipe.uf_init("p_pipe_test", i_src, i_dest, &
   dw_pipe_errors, st_read, st_written, st_errors)
```

uf_init_elapsed_time

Description

Initializes the variable that holds elapsed time information.

Syntax

pipelineinstance.**uf_init_elapsed_time** (*elapsedtime*)

Parameter	Description
pipelineinstance	Instance name of the pipeline user object
elapsedtime	Statictext window control to contain the elapsed time

Return value

None

Usage

Call this user object function to initialize the user object variable that holds elapsed time information.

Example

This example calls the uf_init_elapsed_time function (st_elapsed_time is a window control).

```
i_pipe.uf_init_elapsed_time ( st_elapsed_time )
```

uf_repair

Description

Resubmits corrected rows to the destination database.

Syntax

pipelineinstance.**uf_repair** ( )

Parameter	Description
pipelineinstance	Instance name of the pipeline user object

Return value

Integer. Returns *1* if the Repair function succeeded and *-1* if it failed (or if the pipeline was already executing).

Usage

Call this user object function to resubmit corrected rows to the destination database.

Example

This example calls the uf_repair function.

```
int li_rc

li_rc = i_pipe.uf_repair( )
IF li_rc < 0 THEN
   MessageBox("Pipeline Repair Error", &
      i_pipe.uf_get_error_msg( li_rc ),
Exclamation!)
END IF
```

See also

uf_execute
uf_cancel

uf_set_commit

Description

Sets the Commit frequency.

Syntax

pipelineinstance.**uf_set_commit** (*commitfactor*)

Parameter	Description
pipelineinstance	Instance name of the pipeline user object.
commitfactor	Long specifying the number of rows to be copied between COMMITs. *Commitfactor* must be one of the following: ♦ 0 (commit after all updates) ♦ 1 ♦ 10 ♦ 100 ♦ 1,000 ♦ 10,000 ♦ 100,000

Return value

Boolean. Returns TRUE if the function succeeded and FALSE if it did not.

Usage

Call this user object function to control the frequency with which the pipeline issues database COMMITs.

Example

This example calls the uf_set_commit function.

```
long ll_parm

ll_parm = 1000
IF NOT i_pipe.uf_set_commit(ll_parm) THEN
   MessageBox("Failure","Commit could not be set")
END IF
```

See also

uf_get_commit

uf_set_extended_attr_copy

Description

Controls whether the pipeline copies extended attributes.

Syntax

pipelineinstance.**uf_set_extended_attr_copy** (*copy*)

Parameter	Description
pipelineinstance	Instance name of the pipeline user object
copy	Boolean that specifies whether to copy attributes (TRUE) or not (FALSE)

Return value

Boolean. Returns TRUE if the function succeeded and FALSE if it did not.

Usage

Call this user object function to control whether the pipeline copies extended attributes.

Example

This example calls the uf_set_extended_attr_copy function.

```
boolean  lb_copy
IF cbx_copy_extended.checked = TRUE THEN
   lb_copy = i_pipe.uf_set_extended_attr_copy(TRUE)
ELSE
   lb_copy = i_pipe.uf_set_extended_attr_copy(FALSE)
END IF
```

See also

uf_get_extended_attr_copy

uf_set_maxerrors

Description

Sets the value of the attribute indicating the number of errors the pipeline will allow before canceling execution.

Syntax

pipelineinstance.**uf_set_maxerrors** (*maxerrors*)

Parameter	Description
pipelineinstance	Instance name of the pipeline user object
maxerrors	Long indicating the number of errors the pipeline will allow before canceling execution

Return value Boolean. Returns TRUE if the function succeeded and FALSE if it did not.

Usage Call this user object function to control the maxerrors value.

Example This example calls the uf_set_maxerrors function.

```
long  ll_parm
ll_parm = 100
IF NOT i_pipe.uf_set_maxerrors(ll_parm) THEN
   MessageBox("Failure","MaxErrors could not be
set")
END IF
```

See also uf_get_maxerrors

uf_set_syntax_value

Description Sets a value in the data pipeline object. This is an internal function that is used by other u_pipeline_kit user object functions; do not call this function directly.

Syntax *pipelineinstance*.**uf_set_syntax_value** (*item, value*)

Parameter	Description
pipelineinstance	Instance name of the pipeline user object
item	String specifying the item whose value will be changed
value	String specifying the new value

Return value Boolean. Returns TRUE if the function succeeded and FALSE if it did not.

Usage Call this user object function to modify attributes of the data pipeline object (that is, the object defined in the Data Pipeline painter). For example, you can use this function to change values for source database, destination database, commit frequency, and so on.

Export a pipeline object to view sample syntax
To see an example of the parameters used in pipeline syntax, use the library painter to export a pipeline object to a text file. You can then examine the file using the PowerBuilder File Editor, the Windows Notepad, or some other ASCII text editor.

Example

This example calls the uf_set_syntax_value function to reset the COMMIT frequency to 1000.

```
IF NOT &
   i_pipe.uf_set_syntax_value("commit","1000") THEN
      MessageBox("Commit", &
         "Problem changing Commit factor")
END IF
```

See also

uf_get_syntax_value
uf_get_type
uf_set_type

uf_set_type

Description

Modifies the pipeline processing type (create, replace, append, and so on).

Syntax

pipelineinstance.**uf_set_type** (*pipelinetype*)

Parameter	Description
pipelineinstance	Instance name of the pipeline user object.
pipelinetype	String specifying whether one of the following: ♦ Replace ♦ Create ♦ Update ♦ Refresh ♦ Append

Return value

Boolean. Returns TRUE if the function succeeded and FALSE if it did not.

Usage

Call this user object function to modify the pipeline processing type.

Example

This example calls the uf_set_type function to change the processing type to Append.

```
IF NOT i_pipe.uf_set_type("append") THEN
   MessageBox("Set Type", &
      "Unable to modify processing type.")
END IF
```

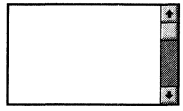
See also

uf_get_syntax_value
uf_get_type
uf_set_syntax_value

uo_dw

Description

Standard user object that is a DataWindow and contains precoded scripts for events, user events, and user object functions.



Library

SYS.PBL

Instance variables

Variable	Data type	Access
ib_allow_updates	Boolean	Public
ib_allow_inserts	Boolean	Public
il_selected_row	Long	Public
isrt_validations[]	Validation_struct	Public

Structures

Validation_struct An array that defines the relationships between rules and their error messages. You add rules to this structure array with the uf_add_validation user object function.

Field	Data type
expression	String

Field	Data type
error_message	String

User object functions

The user object functions encapsulated in uo_dw are:

- Uf_add_validation, explained on page 318.
- Uf_check_required, explained on page 319.
- Uf_is_modified, explained on page 320.
- Uf_validate, explained on page 320.

User object events

Event	Description
Clicked	Sets the current row in the DataWindow to the clicked row even if the user clicks on a column with a tab value of 0.
DBError	Sets focus to the row in error and uses the f_error_box function to display a popup window with an error message.
SQLPreview	Triggers the appropriate event for additional processing to check each update, insert, or delete before actually sending it to the DBMS. If the event determines the action is invalid, it sets Message.Return value to *1* and then returns.
add_row (user event)	Inserts a row at the bottom of the DataWindow and sets focus to that row.
del_row (user event)	Prompts the user with a message box when a row is about to be deleted. If the user's response is Yes, the row is deleted.
del_all_rows (user event)	Deletes all the rows in the DataWindow.
on_delete (user event)	Adds application-specific delete processing to the descendent DataWindow control. This event is triggered in the SQLPreview event just before the SQL to delete the row is sent to the database.
on_insert (user event)	Adds application-specific insert processing to the descendent DataWindow control. This event is triggered in the SQLPreview event just before the SQL to insert the row is sent to the database.
on_update (user event)	Adds application-specific update processing to the descendent DataWindow control. This event is triggered in the SQLPreview event just before the SQL to delete the row is sent to the database.

Usage

Use this user object in windows instead of a DataWindow control.

In Application Library application framework windows, the uo_dw user object is associated with the dw_sheet control.

Example

For examples of uo_dw usage, see any of the application framework sheet windows.

See also

w_sys_mast_detl_dw
w_sys_multi_dw
w_sys_report
w_sys_shared_dw
w_sys_single_dw

uf_add_validation

Description

Adds application-specific validation rule(s) for the DataWindow. These are rules that are applied to all rows that have been modified and can be formulated so that any combination of columns for a row can be combined into one expression.

Syntax

controlname.**uf_add_validation** (*expression, errormessage*)

Parameter	Description
controlname	The window control name of the uo_dw user object.
expression	String used to define the rule. Rules are strings in the same format as validation rules defined in the DataWindow painter. For example, to make sure that the start_date is before the end_date, use the following rule: `start_date < end_date` These rules must evaluate to a boolean result.
errormessage	String containing the error message to be displayed when the rule fails.

Return value

None

Usage

You typically call this function from the Open event of a window to set the validation rule(s) for the control.

Example

This example calls uf_add_validation in the Open event of a window where uo_dw is named dw_sheet.

```
dw_sheet.uf_add_validation("len(trim(au_id)) = 9", &
   "Author id must be 9 characters in length")
```

uf_check_required

Description

Checks to see if there are any columns on the DataWindow marked as required and not filled in. If a required entry has not been entered, it issues an error message and sets focus to the row and column that has not been filled in.

Syntax

controlname.**uf_check_required** ( )

Parameter	Description
controlname	The window control name of the uo_dw user object

Return value

Boolean. Returns TRUE if all required columns have completed and FALSE if they have not.

Usage

Call this user object function to see if there are any columns on the DataWindow that have the Required attribute set to TRUE and have not been filled in.

Example

This example calls the uf_check_required function.

```
// Check for required fields.
IF NOT dw_sheet.uf_check_required() THEN return
FALSE
```

uf_is_modified

Description

Determines if there have been any modifications to the data in the control.

Syntax

controlname.**uf_is_modified** ( )

Parameter	Description
controlname	The window control name of the uo_dw user object.

Return value

Boolean. Returns TRUE if any data has been modified or if rows have been inserted or deleted. Otherwise it returns FALSE.

Usage

Call this user object function in the CloseQuery event of a window to see if any columns have been modified. If unmodified data exists, use the f_exit_status function to ask the user how to proceed.

Example

This example calls the uf_is_modified function.

```
// Are there unsaved changes?
IF NOT dw_sheet.uf_is_modified() THEN return FALSE
```

uf_validate

Description

Test data in the control against previously added rules. You can pass a row number to specify the row that is validated.

Syntax

controlname.**uf_validate** (*rownumber*)

Parameter	Description
controlname	The window control name of the uo_dw user object.
rownumber	Long specifying which row to test. If *rownumber* is 0 then all modified rows are tested.

Return value

Boolean. Returns TRUE if the row passes validation and FALSE if it does not.

Usage

You typically call this function before saving the DataWindow or when the user has moved to a new row in the DataWindow.

Example

This example calls the uf_validate function.

```
// Check for validation errors on the entire
// DataWindow.
IF NOT dw_sheet.uf_validate(0) THEN return FALSE
```

PART FOUR

Sample Applications

This part describes the sample applications and code examples included with the Application Library.

CHAPTER 10

Pubs Sample Application

About this chapter

This chapter describes the Pubs sample application. By running this application and reviewing its events, windows, and functions, you can learn about using the Application Library in a complex application.

Contents

Application setup

Before running the Pubs sample application, you must:

- Add Application Library libraries to the a_pubs application library search path
- Configure the PUBS database
- Modify the PUBS.INI file

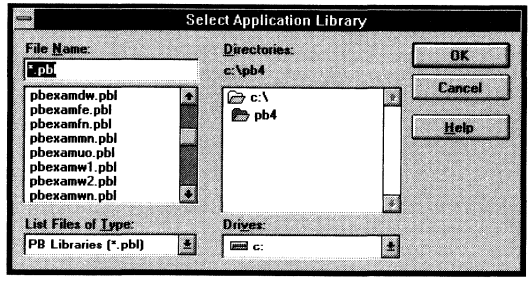
❖ To modify the application library search path:

1 Start PowerBuilder.

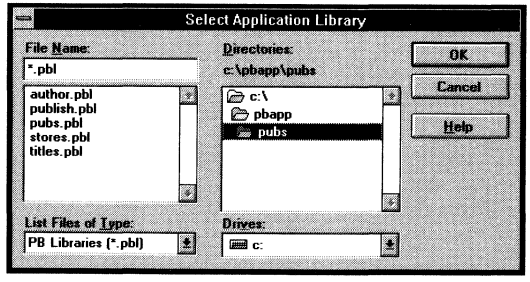
2 Open the Application painter.

3 Select File➢Open from the menu bar.

The Select Application Library dialog box displays.



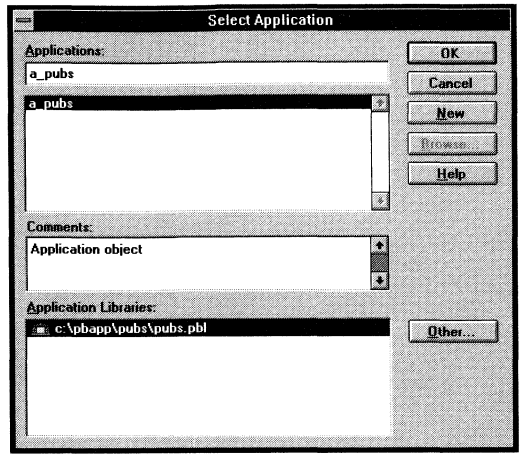
4 Use the Directories listbox to access the PBAPP\PUBS directory.



5 Double-click PUBS.PBL.

6 Click OK.

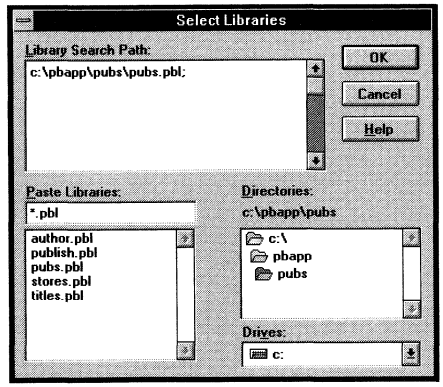
The Select Application dialog box displays.



7 Click OK.

8 Click the Library List button in the PainterBar.

The Select Libraries dialog box displays.



9 Include the following libraries in your library search path by double-clicking on the PBL name in the Paste Libraries listbox. This is the recommended order.

Library	Location
PUBS.PBL	\PBAPP\PUBS (included automatically)
SYS.PBL	\PBAPP
UTLFUNC.PBL	\PBAPP

Library	Location
UTLWIN.PBL	\PBAPP
AUTHOR.PBL	\PBAPP\PUBS
PUBLISH.PBL	\PBAPP\PUBS
STORES.PBL	\PBAPP\PUBS
TITLES.PBL	\PBAPP\PUBS

10 Click OK.

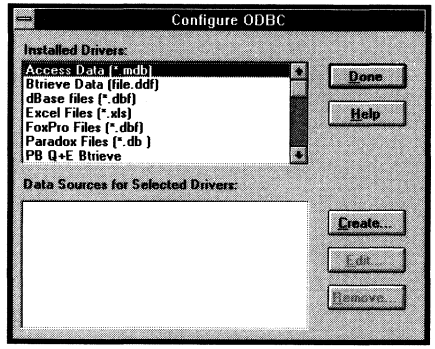
The Application painter workspace displays.

❖ To configure the PUBS database:

1 Open the Database painter.

2 Select File➢Configure ODBC from the menu bar.

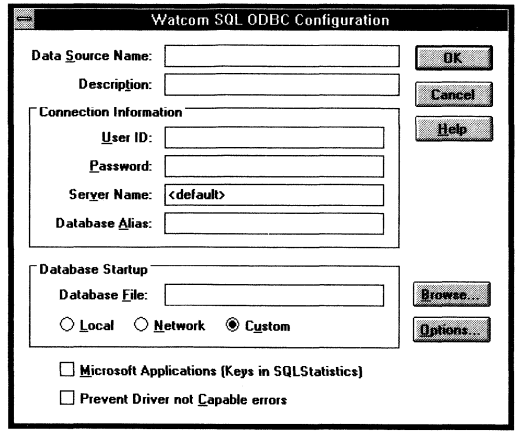
The Configure ODBC dialog box displays.



3 Select Watcom SQL 4.0 from the list of installed drivers.

4 Click Create.

The WATCOM SQL ODBC Configuration dialog box displays.



5 Enter this information:

Field	Value
Data Source Name	pubs
Description	Pubs Database
User ID	dba
Password	sql
Database file	*pathname*\pubs.db This is typically C:\PBAPP\PUBS\PUBS.DB
Database Startup	Select Local

6 Click OK.

The Configure ODBC dialog box displays.

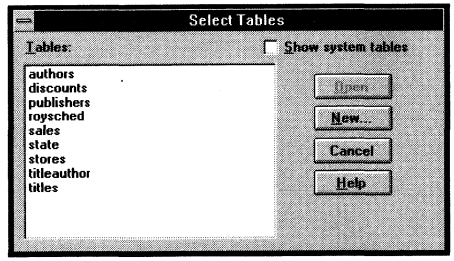
7 Click Done.

The Database painter workspace displays.

8 Select File➤Connect from the menu bar.

9 Select pubs from the list of profiles.

The list of tables displays.



10 Click Cancel.

These steps register the PUBS database with PowerBuilder so that you can run the Pubs sample application.

❖ To modify the PUBS.INI file:

1 Using the PowerBuilder File Editor, the Windows Notepad, or any other ASCII text editor, open the PUBS.INI file in the PBAPP\PUBS directory.

2 Replace the bracketed names with your own settings.

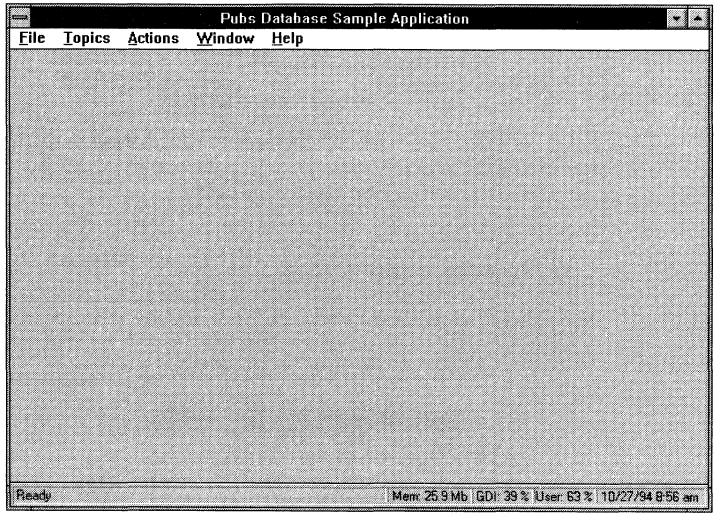
```
; Setup for ODBC
DBMS=ODBC
ServerName=
Database=
UserId=
DbParm=Connectstring='DSN=PUBS;UID=DBA;PWD=SQL'
```

> **For ODBC, use DBMS and DbParm only**
> For ODBC, you only need to specify values for DBMS and DbParm. For other databases, you may need to specify additional values. To determine the values to include in the PUBS.INI file, refer to the Pubs database Profile section in the PB.INI file.

3 Save the file.

Usage instructions

When you run the Pubs sample application, the w_login window displays and, after a successful login, the frame window displays.



From the frame window, you have many options:

- Display and modify author information
- Display and modify publisher information
- Display and modify store information
- Display and modify title information

Not all functionality is covered
The following discussions provide an overview of what you can do with the Pubs sample application. They do not cover every possible action. For example, they do not cover updating and deleting information, which you perform using Save and Delete items on the File menu.

Accessing author information

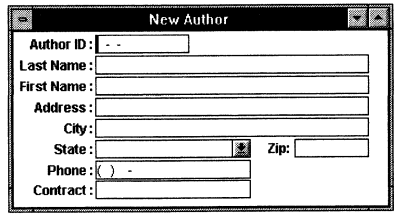
Tasks you can perform related to authors include:

- Add an new author
- Access information for an existing author
- Access titles for an author
- Add titles for an author
- Access author Master/Detail information for published titles
- Delete a title for an author

❖ To add a new author:

1 Select Topics➢Authors from the menu bar.

The New Author window displays.



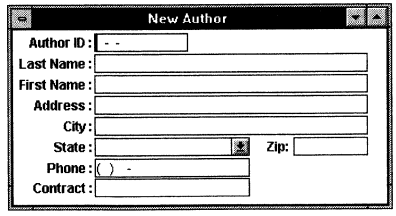
2 Add new author information.

3 Click the save button.
or
Select File➢Save from the menu bar.

❖ To access information for an existing author:

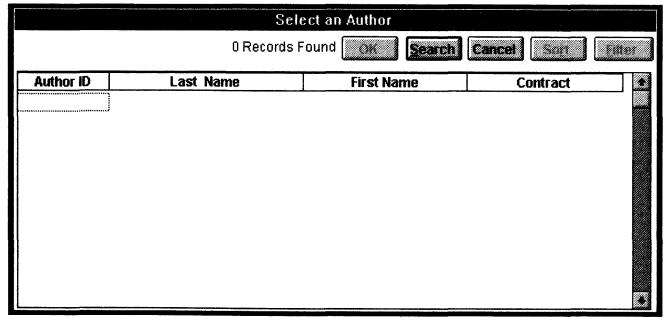
1 Select Topics➢Authors from the menu bar.

The New Author window displays.

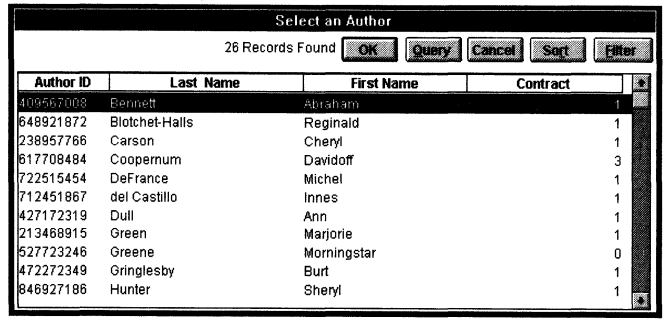


2 Click the Open button.
or
Select File➤Open from the menu bar.

The Select an Author dialog box displays.



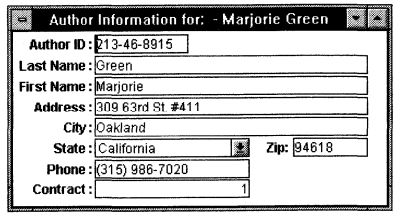
3 Display the author list by clicking Search. The text for the button toggles to Query.



You can also use Query mode
Because this dialog box is a descendant of w_select, you can also use query mode to display the author list. To use query mode, type an author ID, last name, first name, or contract into the first line of an empty window and click Search. The author list displays rows that match the specified query. To return to query mode from the author list display, click Query.

4 Select an author by double-clicking or by clicking the author and clicking OK.

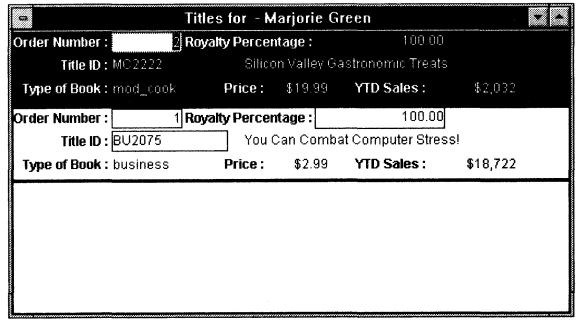
The Author Information window displays.



❖ To access titles for an author:

- With the Author Information window displayed, select Actions➤Titles for Author from the menu bar.

  The Titles for Author window displays.



❖ To add titles for an author using this window:

1. With the Titles for Author window displayed, select Actions➤Insert Row from the menu bar.

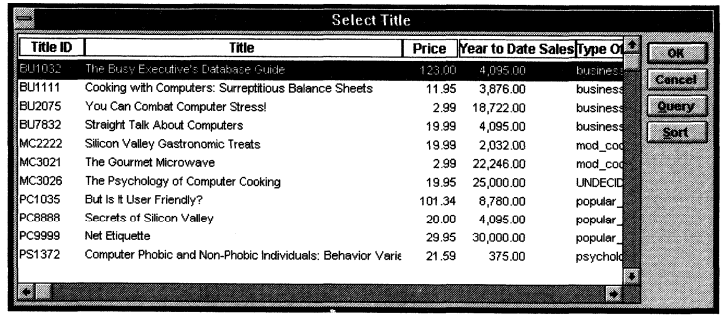
An empty row displays.

2 Position the cursor over the Title ID column.

The cursor changes to an up arrow, which indicates that you can display a selection list by double-clicking.

3 Double-click.

The Select Title dialog box displays.



You can also use query mode with this dialog box. For information on using query mode, see "Accessing author information" on page 332.

4 Double-click on the desired title.

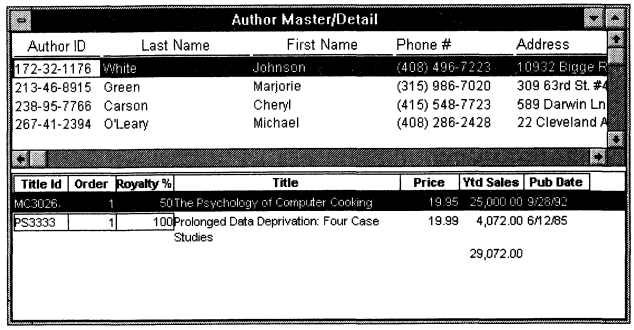
The Titles for Author window displays with the selected title in the new row.

5 Click the Save button.
or
Select File➤Save from the menu bar.

❖ To access author Master/Detail information:

- Select Topics➤Author Master/Detail from the menu bar.

The Author Master/Detail window displays.



❖ To add a new title for an author using the Master/Detail window:

1 Select the desired author in the top half of the window.

2 Click the Insert Detail button.
or
Select Actions Insert Detail from the menu bar.

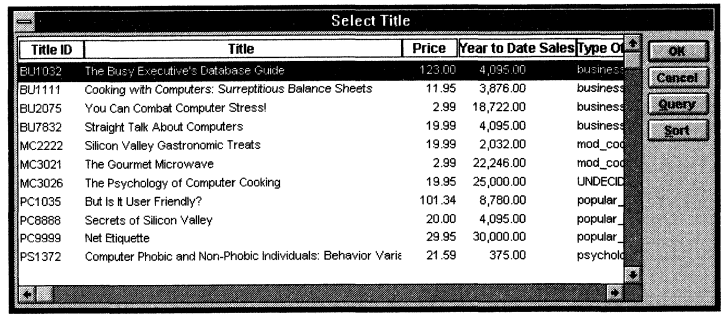
A blank row displays in the bottom half of the window.

3 Position the cursor over the Title ID column.

The cursor changes to an up arrow, which indicates that you can display a selection list by double-clicking.

4 Double-click.

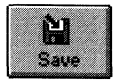
The Select Title dialog box displays.



You can also use query mode with this dialog box. For information on using query mode, see "Accessing author information" on page 332.

5 Double-click on the desired title.

The Author Master/Detail window displays with the selected title in the new row.



6 Click the Save button.
or
Select File➤Save from the menu bar.

❖ To delete a title for an author:

1 In the Author Master/Detail window, click in the title to be deleted.

2 Click the Delete button.
or
Select Actions➤Delete Detail from the menu bar.

Accessing publisher information

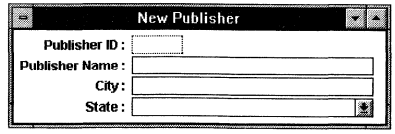
Tasks you can perform related to publishers include:

- Add an new publisher
- Access information for an existing publisher
- Access titles for a publisher

❖ To add a new publisher:

1 Select Topics➤Publishers from the menu bar.

The New Publisher window displays.



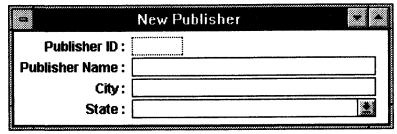
2 Add new publisher information.

3 Click the save button.
or
Select File➤Save from the menu bar.

❖ To access information for an existing publisher:

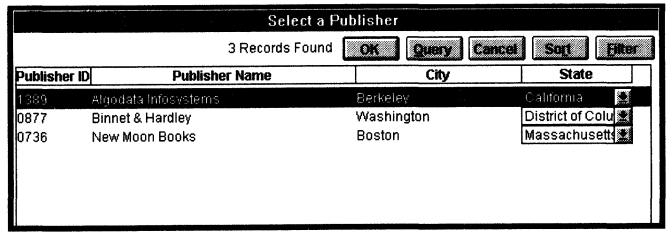
1 Select Topics➤Publishers from the menu bar.

The New Publisher window displays.



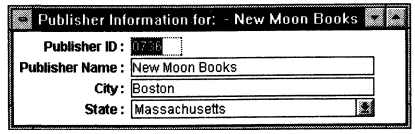
2 Click the Open button.
or
Select File➤Open from the menu bar.

The Select a Publisher dialog box displays. Since there are relatively few publishers, the application displays this list immediately.



4 Select a publisher by double-clicking or by clicking the publisher and clicking OK.

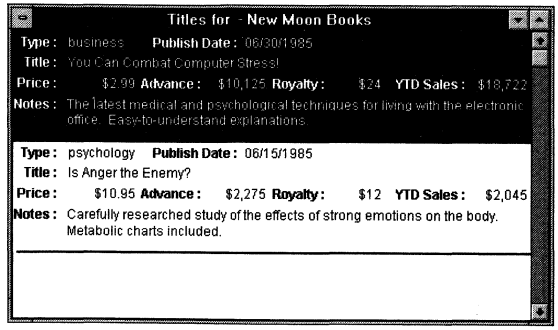
The Publisher Information window displays.



❖ To access titles for a publisher:

- With the Publisher Information window displayed, select Actions➤Titles for Publisher from the menu bar.

The Titles for Publisher dialog box displays.



Accessing store information

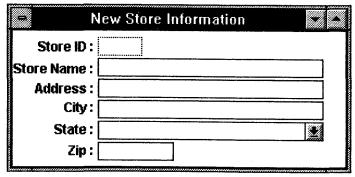
Tasks you can perform related to stores include:

- Add an new store
- Access information for an existing store
- Access sales for a store
- Access discounts for a store

❖ To add a new store:

1 Select Topics➢Stores from the menu bar.

The New Store Information window displays.



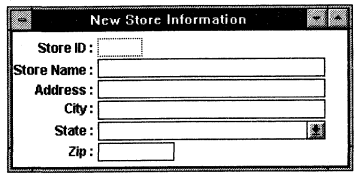
2 Add new store information.

3 Click the save button.
or
Select File➢Save from the menu bar.

❖ To access information for an existing store:

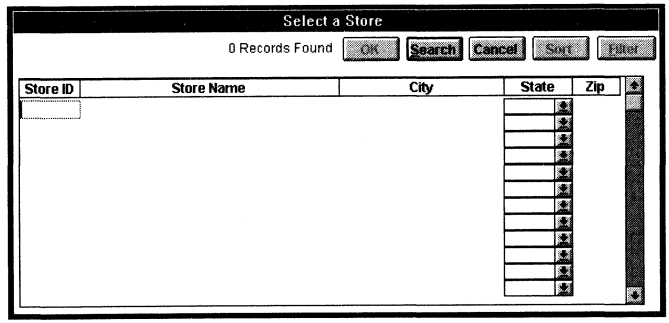
1 Select Topics➤Stores from the menu bar.

The New Store Information window displays.



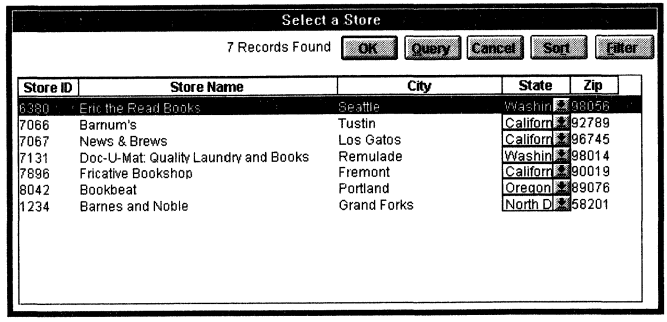
2 Click the Open button.
or
Select File➤Open from the menu bar.

The Select a Store dialog box displays.



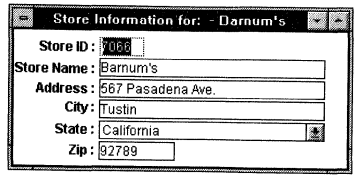
You can also use query mode with this dialog box. For information on using query mode, see "Accessing author information" on page 332.

3 Display the store list by clicking Search.



4 Select a store by double-clicking or by clicking the store and clicking OK.

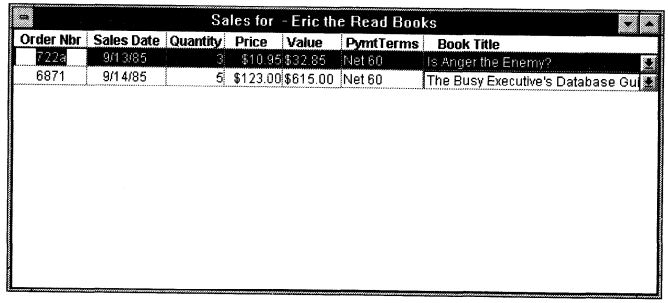
The Store Information window displays.



❖ To access sales for a store:

1 With the Store Information window displayed, select Actions➢Sales for Store from the menu bar.

The Sales for Store window displays.

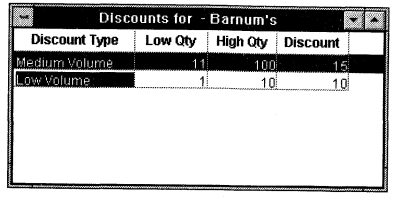


2 With this window displayed, you can use the Actions menu to insert, delete, and sort rows.

❖ To access discounts for a store:

1 With the Store Information window displayed, select Actions➢Discounts for Store from the menu bar.

The Discounts for Store window displays.



2 With this window displayed, you can use the Actions menu to insert, delete, and sort rows.

Accessing title information

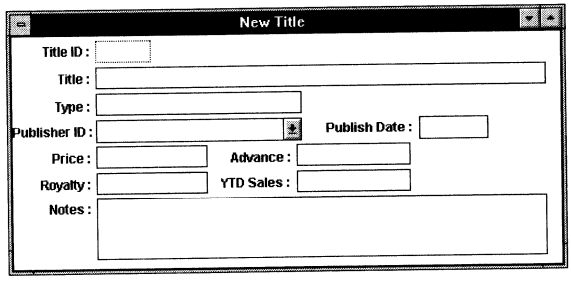
Tasks you can perform related to titles include:

- Add a new title
- Access information for an existing title
- Access sales for a title
- Access royalty schedules for a title

❖ To add a new title:

1 Select Topics➢Titles from the menu bar.

The New Title window displays.



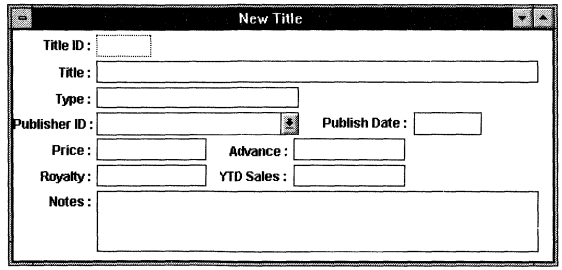
2 Add new title information.

3 Click the save button.
or
Select File➢Save from the menu bar.

❖ To access information for an existing title:

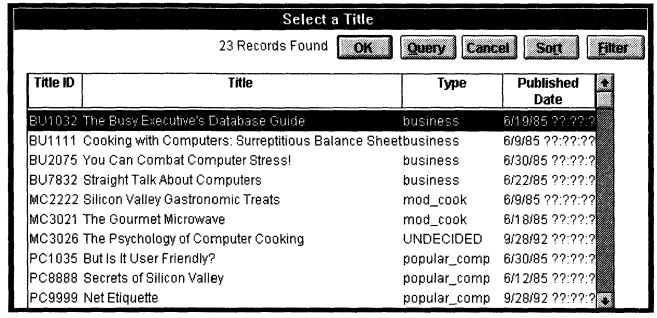
1 Select Topics➢Titles from the menu bar.

The New Title window displays.



2 Click the Open button.
or
Select File➢Open from the menu bar.

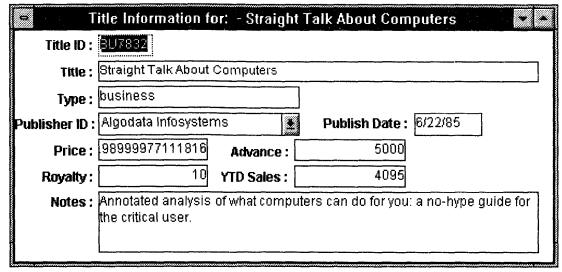
The Select a Title dialog box displays.



You can also use query mode with this dialog box. For information on using query mode, see "Accessing author information" on page 332.

3 Select a title by double-clicking or by clicking the title and clicking OK.

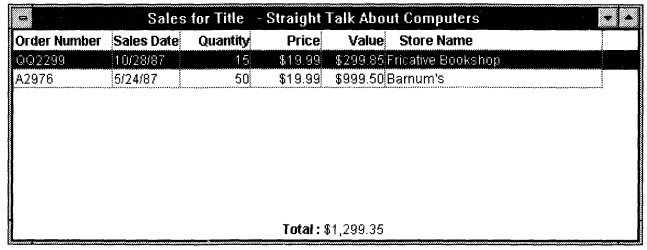
The Title Information window displays.



❖ To access sales for a title:

- With the Title Information window displayed, select Actions➢Sales for Title from the menu bar.

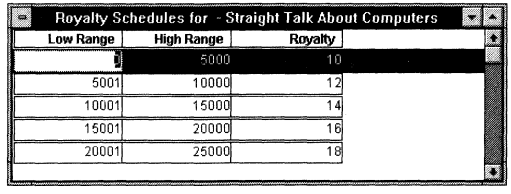
  The Sales for Title window displays.



❖ To access the royalty schedule for an existing title:

1. With the Title Information window displayed, select Actions➢Royalty Schedules from the menu bar.

   The Royalty Schedules window displays.



2. With this window displayed, you can use the Actions menu to insert, delete, and sort rows.

Things to note

Selection windows

The Pubs application uses descendants of the w_select window to select rows to add to a DataWindow. These windows are:

- w_author_select
- w_publishers_select
- w_store_select
- w_title_select

Control of window display

The Pubs application keeps track of the contents of each sheet and will allow only one sheet instance for each row. That is, for a certain title, the Pubs application will allow only one instance of the Title Information window for that title. This is accomplished by calling the wf_set_sheettitle window function in the ue_fileopen user event of the sheet window. The wf_set_sheettitle window function is defined in the w_sys_single_dw ancestor window.

Application Library object usage

This discussion outlines some of the Application Library objects used in the Pubs application

- **m_sys_frame menu**
  - m_pubs menu
    - m_author_master_detail menu
    - m_author_sheet menu
    - m_discount_sheet menu
    - m_publishers_menu menu
    - m_sales_sheet menu
    - m_store_sheet menu
    - m_title_roysched menu
    - m_title_sales menu
    - m_title_sheet menu
    - m_titleauthor_sheet menu
    - m_titlepubs_sheet menu

- **w_select window**
  - w_author_select window
  - w_publishers_select window
  - w_store_select window
  - w_title_select window
- **w_sys_frame window**
  - w_pubs_frame window
- **w_sys_mast_detl_dw window**
  - w_authors_master_detail window
- **w_sys_single_dw window**
  - w_author_sheet window
  - w_publishers_sheet window
  - w_store_sheet window
- **w_sys_multi_dw window**
  - w_publishers_title_sheet window
  - w_store_discounts_sheet window
  - w_store_sales_sheet window
  - w_title_roysched window
  - w_title_sales window
- **w_sort_order window**
  - Opened any time you choose the Sort option

CHAPTER 11

Time Management Sample Application

About this chapter

This chapter describes the Time Management sample application. By running this application and reviewing its events, windows, and functions, you can learn about using the Application Library in a complex application.

Contents

Application setup

Before running the Time Management sample application, you must:

- Add Application Library libraries to the TIMEMGMT application library search path
- Configure the TIMEMGMT database
- Modify the TIMEMGMT.INI file

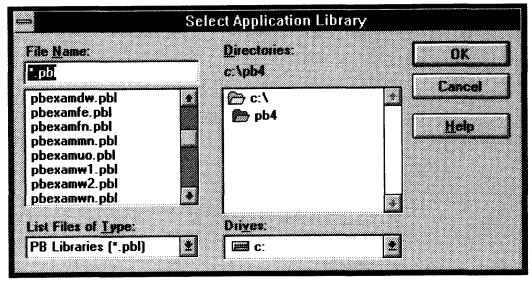
❖ To modify the application library search path:

1 Start PowerBuilder.

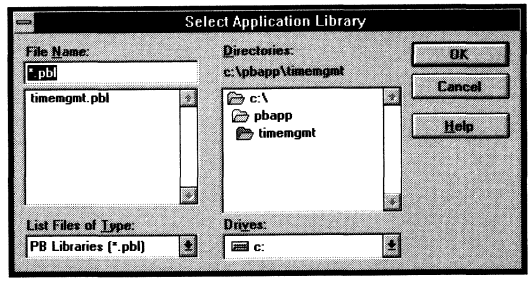
2 Open the Application painter.

3 Select File➢Open from the menu bar.

The Select Application Library dialog box displays.



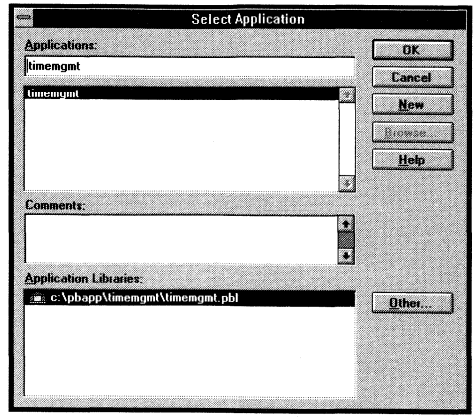
4 Use the Directories listbox to access the PBAPP\TIMEMGMT directory.



5 Double-click TIMEMGMT.PBL.

6 Click OK.

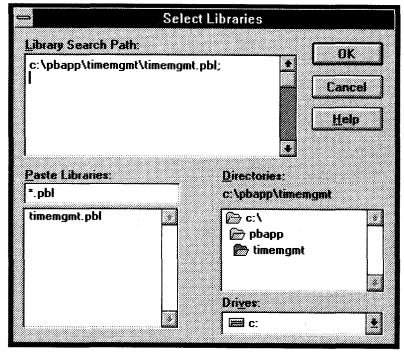
The Select Application dialog box displays.



7 Click OK.

8 Click the Library List button in the PainterBar.

The Select Libraries dialog box displays.



9 Include the following libraries in your library search path by double-clicking on the PBL name in the Paste Libraries listbox. This is the recommended order.

Library	Location
TIMEMGMT.PBL	\PBAPP\TIMEMGMT (included automatically)
SYS.PBL	\PBAPP
UTLFUNC.PBL	\PBAPP
UTLWIN.PBL	\PBAPP

10 Click OK.

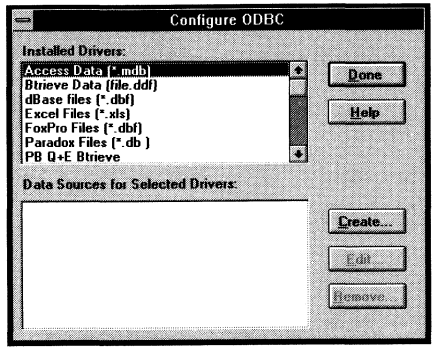
The Application painter workspace displays.

❖ To configure the TIMEMGMT database:

1 Open the Database painter.

2 Select File➤Configure ODBC from the menu bar.

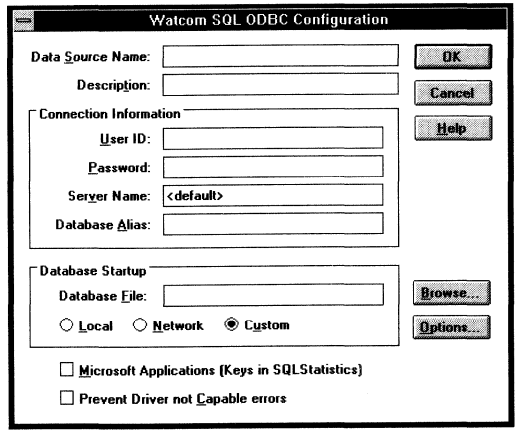
The Configure ODBC dialog box displays.



3 Select Watcom SQL 4.0 from the list of installed drivers.

4 Click Create.

The WATCOM SQL ODBC Configuration dialog box displays.



5 Enter this information:

Field	Value
Data Source Name	timemgmt
Description	Time Management Database
User ID	dba
Password	sql
Database file	*pathname*\timemgmt.db This is typically C:\PBAPP\TIMEMGMT\TIMEMGMT.DB
Database Startup	Select Local

6 Click OK.

The Configure ODBC dialog box displays.

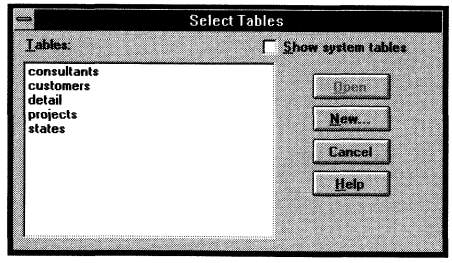
7 Click Done.

The Database painter workspace displays.

8 Select File➤Connect from the menu bar.

9 Select timemgmt from the list of profiles.

The list of tables displays.



10.Click Cancel.

These steps register the TIMEMGMT database with PowerBuilder so that you can run the Time Management sample application.

❖ To modify the TIMEMGMT.INI file:

1 Using the PowerBuilder File Editor, the Windows Notepad, or any other ASCII text editor, open the TIMEMGMT.INI file in the PBAPP\TIMEMGMT directory.

2 Replace the bracketed names with your own settings.

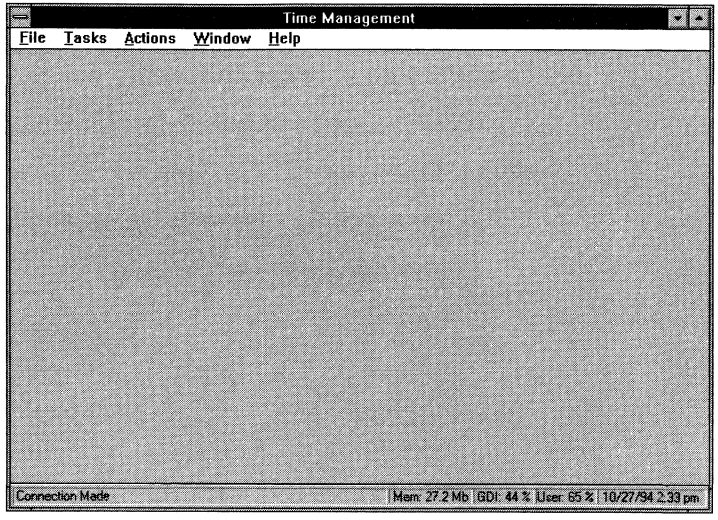
```
; Setup for ODBC
DBMS=ODBC
ServerName=
Database=
UserId=
DbParm=Connectstring='DSN=TIMEMGMT;UID=DBA;PWD=SQL'
```

For ODBC, use DBMS and DbParm only
For ODBC, you only need to specify values for DBMS and DbParm. For other databases, you may need to specify additional values. To determine the values to include in the TIMEMGMT.INI file, refer to the Timemgmt database Profile section in the PB.INI file.

3 Save the file.

Usage instructions

When you run the Time Management sample application, the w_login window displays and, after a successful login, the frame window displays.



From the frame window, you have many options:

- Display and modify consultant information
- Display and modify customer information
- Display and modify state information
- Display reports

Not all functionality is covered
The following discussions provide an overview of what you can do with the Time Management application. They do not cover every possible action. For example, they do not cover updating and deleting information, which you perform using the Save and Delete items on the File menu.

Accessing consultant information

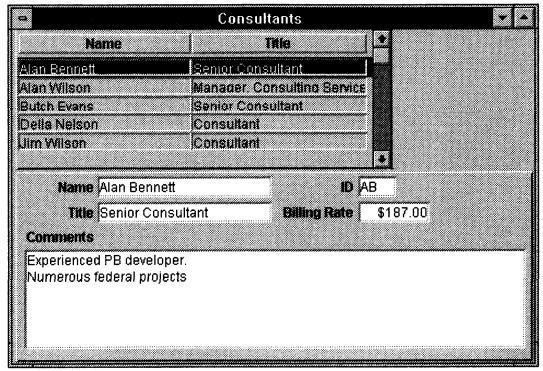
Tasks you can perform related to consultants include:

- Display a list of consultants
- Add a new consultant

❖ To display a list of consultants:

- Select Tasks➤Consultants from the menu bar.

  The Consultants window displays.



❖ To add a new consultant:

1. With the Consultants window displayed, click the New button.
   or
   Select File➤New from the menu bar.

   A blank row displays in both the top and the bottom DataWindow.
2. Add new consultant information to the bottom DataWindow.

3. Click the Save button.
   or
   Select File➤Save from the menu bar.

Accessing customer information

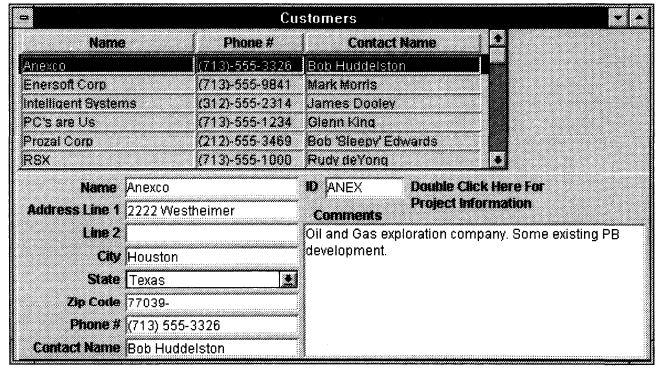
Tasks you can perform related to customers include:

- Display a list of customers
- Add a new customer
- Display project detail
- Generate an invoice
- Display a customer detail report

❖ To display a list of customers:

- Select Tasks➤Customers from the menu bar.

  The Customers window displays.



❖ To add a new customer:

1 With the Customers window displayed, click the New button.
or
Select File➤New from the menu bar.

A blank row displays in both the top and the bottom DataWindow.

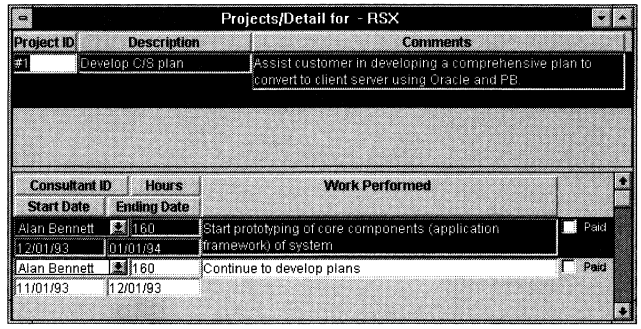
2 Add new customer information to the bottom DataWindow.

3 Click the Save button.
or
Select File➤Save from the menu bar.

❖ To display project detail for a customer:

1 With the Customers window displayed, select Actions➤Project/Detail from the menu bar.

The Project/Detail window displays.

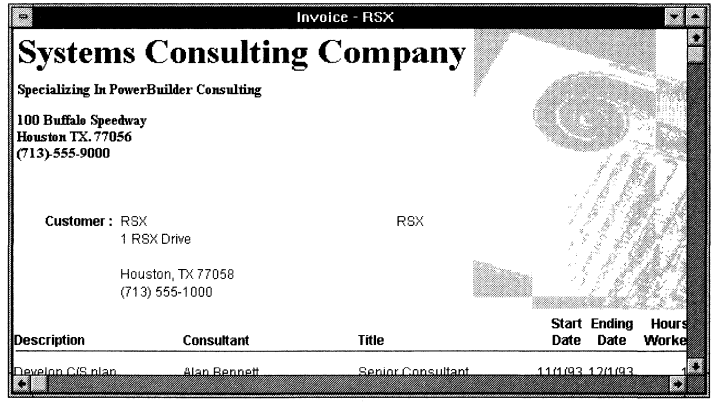


2 With this window displayed, you can use the Actions menu to insert, delete, and sort detail rows.

❖ To generate an invoice for a customer:

1 With the Customers window displayed, select Actions➤Generate Invoice from the menu bar.

The Invoice window displays.

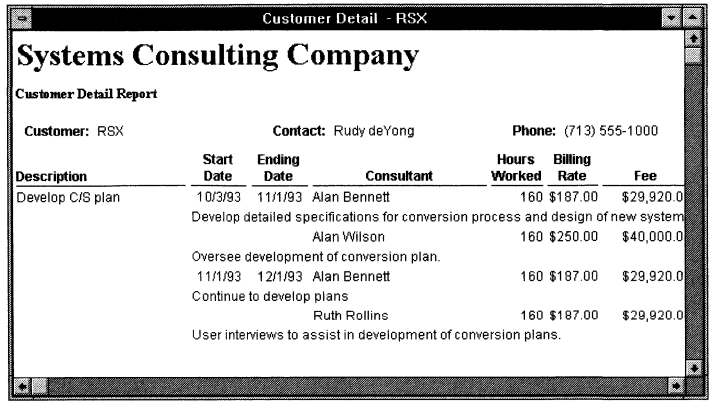


2 At this point, you can view the report online, zoom in, zoom out, and print, optionally selecting File➤Print Preview from the menu bar.

❖ To display a customer detail report:

1 With the Customers window displayed, select Actions➢Detail Report from the menu bar.

The Customer Detail window displays.



2 At this point, you can view the report online, zoom in, zoom out, and print, optionally selecting File➢Print Preview from the menu bar.

Accessing state information

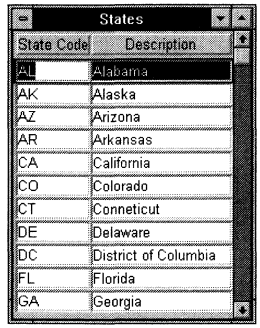
Tasks you can perform related to states include:

- Display state information
- Add a new state

❖ To display state information:

- Select Tasks➢States from the menu bar.

The States window displays.



❖ To add a new state:

1 With the States window displayed, click the New button.
or
Select File➢New from the menu bar.

2 Add new state information.

3 Click the Save button.
or
Select File➢Save from the menu bar.

Accessing reports

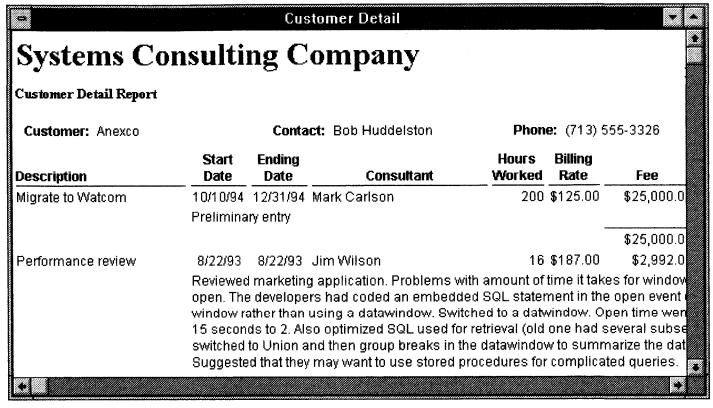
The Time Management reports are:

- Customer detail
- Customer summary
- Consultant detail
- Consultant summary
- Accounts receivable summary
- Accounts receivable aged
- Sales by month

❖ To display the customer detail report:

- Select Tasks➤Reports➤Customer Detail from the menu bar.

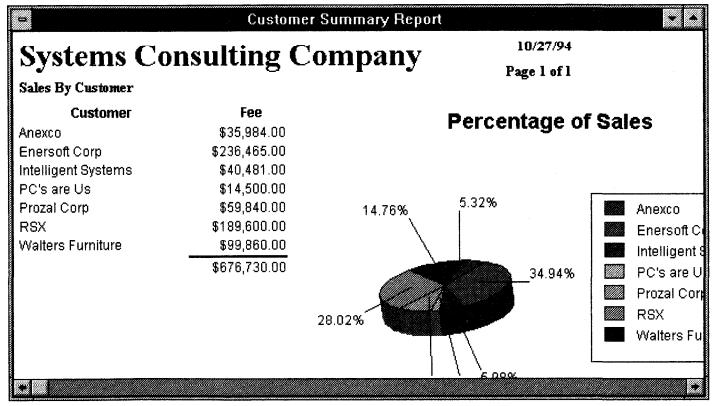
  The Customer Detail report displays. This is the same report as that accessed from the Customer window, but for all customers.



♦ To display the customer summary report:

- Select Tasks➤Reports➤Customer Summary from the menu bar.

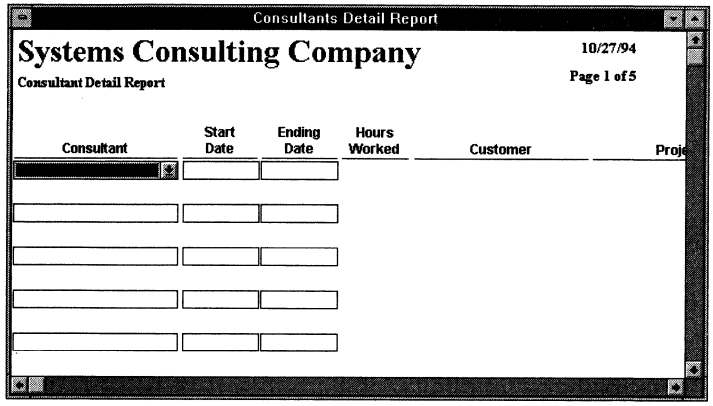
  The Customer Summary report displays.



♦ To display the consultant detail report:

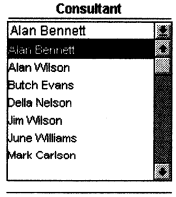
1 Select Tasks➤Reports➤Consultant Detail from the menu bar.

The Consultant Detail query mode window displays.



2 Click on the down arrow at the right of the Consultant column.

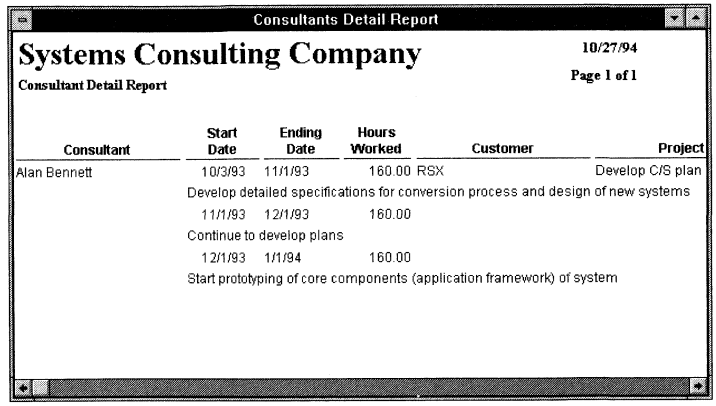
A dropdown listbox displays.



3 Click on the desired consultant.

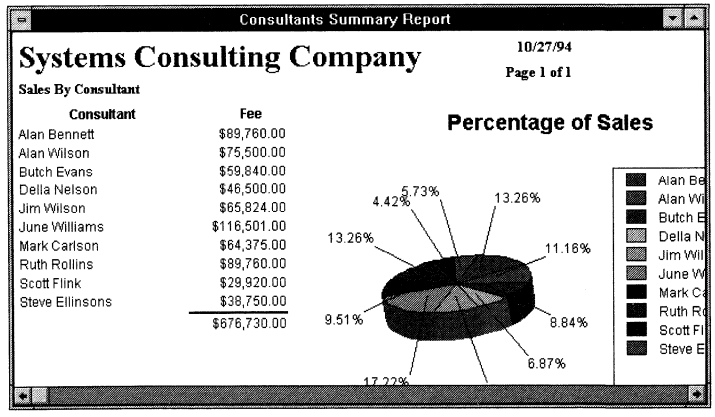
4 Click the Query Mode button.
or
Select File➤Query Mode from the menu bar.

The Consultant Detail report displays for the selected consultant.



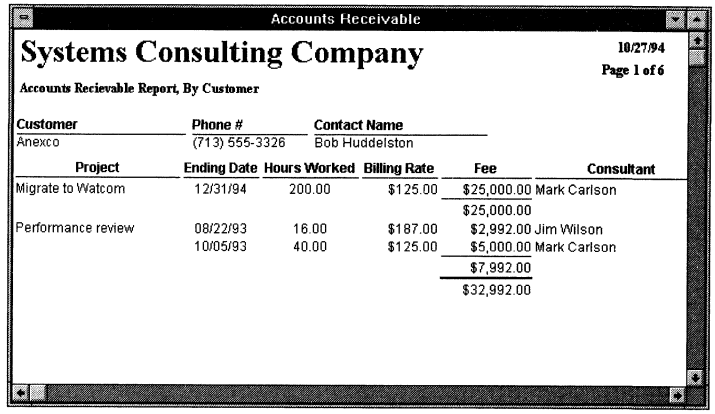
- **To display the consultant summary report:**
  - Select Tasks➢Reports➢Consultant Summary from the menu bar.

The Consultant Summary report displays.



- **To display the accounts receivable summary report:**
  - Select Tasks➢Reports➢Accounts Receivable Detail from the menu bar.

    The Accounts Receivable report displays.



Accounts Receivable

Systems Consulting Company 10/27/94

Page 1 of 6

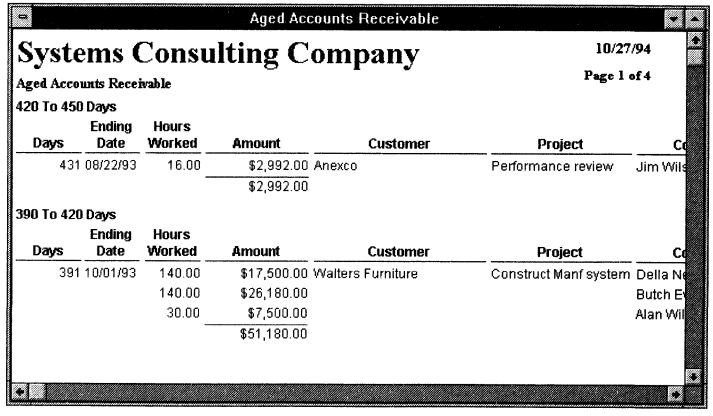
Accounts Recievable Report, By Customer

Customer	Phone #	Contact Name
Anexco	(713) 555-3326	Bob Huddelston

Project	Ending Date	Hours Worked	Billing Rate	Fee	Consultant
Migrate to Watcom	12/31/94	200.00	$125.00	$25,000.00	Mark Carlson
				$25,000.00	
Performance review	08/22/93	16.00	$187.00	$2,992.00	Jim Wilson
	10/05/93	40.00	$125.00	$5,000.00	Mark Carlson
				$7,992.00	
				$32,992.00	

- **To display the accounts receivable aged report:**
  - Select Tasks➢Reports➢Accounts Receivable Aged from the menu bar.

    The Aged Accounts Receivable report displays.



Aged Accounts Receivable

Systems Consulting Company 10/27/94

Page 1 of 4

Aged Accounts Receivable

420 To 450 Days

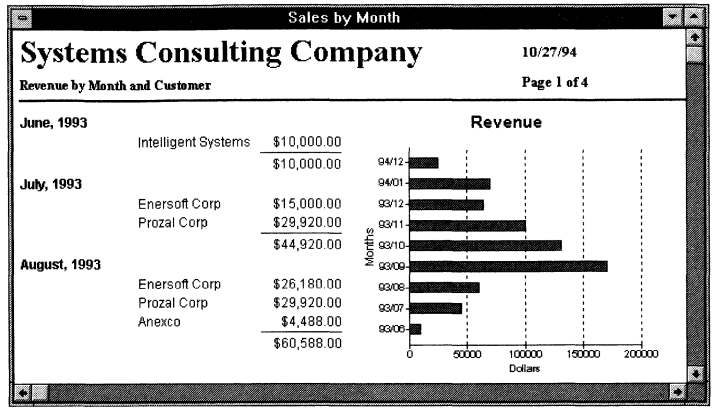
Days	Ending Date	Hours Worked	Amount	Customer	Project	Co
431	08/22/93	16.00	$2,992.00	Anexco	Performance review	Jim Wils
			$2,992.00			

390 To 420 Days

Days	Ending Date	Hours Worked	Amount	Customer	Project	Co
391	10/01/93	140.00	$17,500.00	Walters Furniture	Construct Manf system	Della Ne
		140.00	$26,180.00			Butch E
		30.00	$7,500.00			Alan Wil
			$51,180.00			

- **To display the sales by month report:**
  - Select Tasks➢Reports➢Sales by Month from the menu bar.

    The Sales by Month report displays.



Things to note

Shared DataWindow ancestor

The w_consultants and w_customers windows are descendants of the w_sys_shared_dw shared DataWindow ancestor. Examine these windows for examples of how to use the shared DataWindow feature.

Report ancestor

The report windows are all descendants of the w_sys_report ancestor. Examine these windows for examples of how to use this ancestor. It includes user events for zoom in, zoom out, print preview, and query mode.

Query mode

The w_consultant_summary window uses query mode, allowing you to restrict the report to a single consultant. Examine this window for an example of how to use query mode in a window.

Application Library usage

This discussion outlines some of the Application Library objects used in the Time Management application:

- **m_sys_frame menu**
  - m_timemgmt menu
    - m_consultant menu
    - m_customer menu
    - m_details menu
    - m_report_menu menu
    - m_state menu
- **w_sys_frame window**
  - w_timemgmt_frame window
- **w_sys_mast_detl_dw window**
  - w_details window
- **w_sys_report window**
  - w_rpt_accts_rec window
  - w_rpt_accts_rec_aged window
  - w_rpt_consultant_detail window
  - w_rpt_consultant_summary window
  - w_rpt_customer_detail window
  - w_rpt_customer_summary window

- w_rpt_invoice window
- w_rpt_revenue window

- **w_sys_shared_dw window**
  - w_consultants window
  - w_customers window
- **w_sort window**
  - Opened any time you choose the Sort option

CHAPTER 12

Application Library Code Examples

About this chapter

This chapter describes how to install the code examples delivered with the Application Library. This application (EXAMPLE.PBL in C:\PBAPP\EXAMPLE) demonstrates the usage of the Application Library objects that are not referenced in the sample applications.

The example application allows you to execute a series of windows that show working examples of some of the Application Library's utility functions and objects.

Contents

Application setup

Before running the code examples, you must add Application Library libraries to the example application library search path

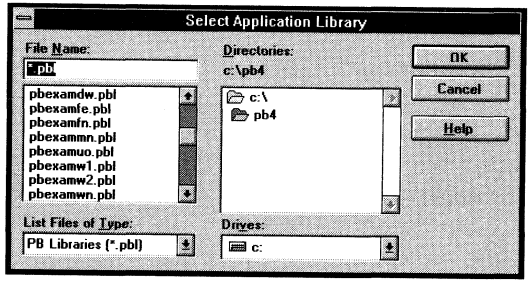
❖ To modify the application library search path:

1 Start PowerBuilder.

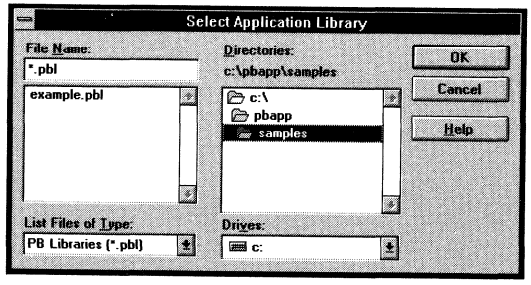
2 Open the Application painter.

3 Select File➤Open from the menu bar.

The Select Application Library dialog box displays.



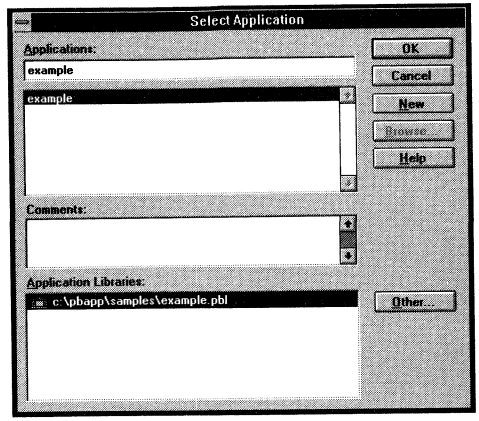
4 Use the Directories listbox to access the PBAPP\EXAMPLE directory.



5 Double-click EXAMPLE.PBL.

6 Click OK.

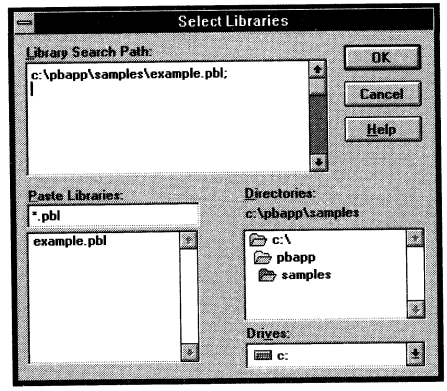
The Select Application dialog box displays.



7 Click OK.

8 Click the Library List button in the PainterBar.

The Select Libraries dialog box displays.



9 Include the following libraries in your library search path by double-clicking on the .PBL name in the Paste Libraries listbox.

Library	Location
EXAMPLE.PBL	\PBAPP\EXAMPLE (included automatically)
SYS.PBL	\PBAPP
UTLFUNC.PBL	\PBAPP
UTLWIN.PBL	\PBAPP

10 Click OK.

The Application painter workspace displays.

❖ To run the Application Library code examples:

1 With example as the current application, select File➤Run from the menu bar.

2 Click the button that correspond to the function you want to see.

APPENDIX

Application INI File

Database connection information

The PowerBuilder Application Library application framework (specifically, the f_app_open function) uses an INI file as the source of database connection information. You must save the application INI file in the same directory as the PBL file that contains the application object.

If desired, you can use the f_app_open function apart from the application framework.

The INI files shown in this manual (PUBS.INI, TIMEMGMT.INI, and TUTOR_AL.INI) all contain parameters for connecting through ODBC to the Watcom SQL database.

Prototype INI file

Modify the application INI file according to your specific application requirements. For ODBC, you only need to specify values for DBMS and DbParm. For other databases, you may need to specify additional values. To determine the values to include for your DBMS, refer to the database's Profile section in the PB.INI file.

Below is a prototype INI file.

```
; Setup for ODBC
[Database1]
DBMS=ODBC
LogId=
LogPassword=
ServerName=
Database=PUBS
UserId=dba
DatabasePassword=
DbParm=Connectstring='DSN=PUBS;UID=DBA;PWD=SQL'
```

Index

E

F

G

H

I

L

M

O

P

Q

R

S

T

U

V

W