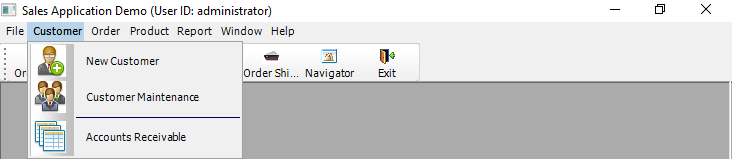
How to Replace an Application Menu with a RibbonBar

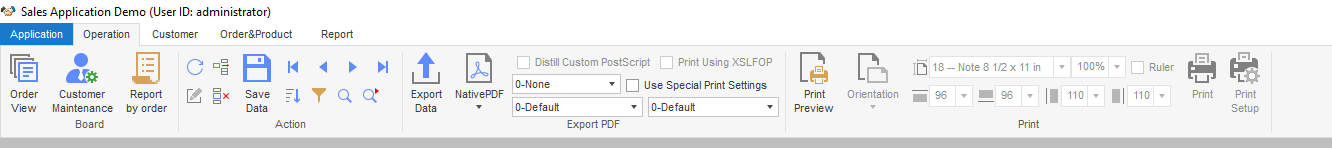
# Overview

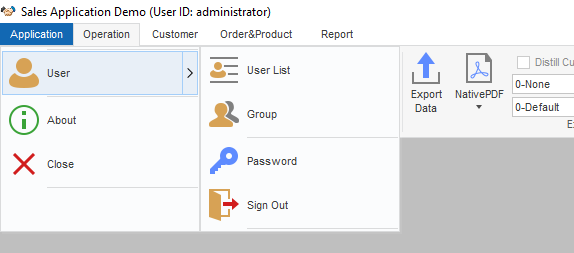
The new RibbonBar control allows you to organize the navigation of your application in a simple, structured way, and gives the application the look and feel of modern applications. Because RibbonBar is provided in PowerBuilder as a control while Menu is a system object, the way that a RibbonBar works in an application is different from the application menu. This tutorial uses a demo application to walk you through the steps of replacing an application menu with a RibbonBar.

The figure below shows what the menu looks like in the original demo application.



The figure below shows what the RibbonBar control looks like after you use it to replace the menu in the application.





This tutorial contains the following sections:

* [Prerequisites](#_Prerequisites)

This explains how to set up the demo application used in the tutorial.

* [Design and create a RibbonBar](#_Initiate_the_RibbonBar)

You shall have a clear understanding of your application to know how to design and create a RibbonBar for it.

* [Add the RibbonBar to the application](#_Associate_functions_with)

The RibbonBar will replace the existing menu in the application. Therefore, you must disable the existing menu before adding the RibbonBar.

* [Associating functions with RibbonBar items](#_Associate_functions_with)

Previously the application functions are all associated with menu items. You need to decide which function to associate with which RibbonBar item.

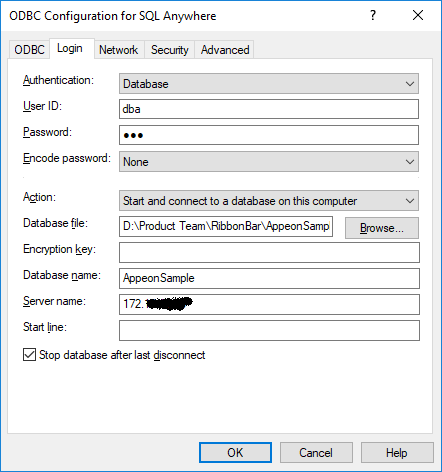
* [Summary notes](#_Important_notes)

Finally, this section highlights the major points you should consider when you start to replace the menu in your application with RibbonBar.

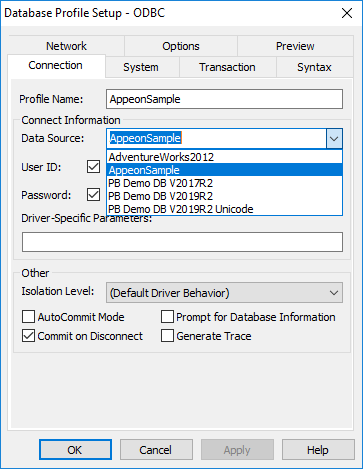
# Prerequisites

* PowerBuilder 2019 R2
* Download the RibbonBar demo application from <https://github.com/Appeon/PowerBuilder-RibbonBar-Example>.
* Sample database setup:
  1. Setup ODBC database connection

1. Open Control Panel > Administrative Tools > ODBC Data Sources (64-bit)
2. Create a new Data Source in SQL Anywhere 17
3. Fill in the data source settings. You can get the database file “AppeonSample.db” in the downloaded source files.The username and password for data source connection is ‘dba/sql’



1. In PowerBuilder, setup and connect to the created ODBC data source



# Design and create a RibbonBar

As the first step, you shall design the RibbonBar, that is, deciding what items to contain in the RibbonBar, what modern-looking image to use for each item, and how to group the items, etc.

Once you know clearly how you want the RibbonBar to look like, you can start to create it. We recommend you use the RibbonBar Builder tool to create the RibbonBar XML file. Alternatively, you can choose to directly create a RibbonBar via PowerScript.

For detailed instructions on how to create a RibbonBar, refer to PowerBuilder Help > RibbonBar control. In the Sales Demo application, for better illustration purpose, you will see both the XML file and PowerScript code are provided for create the RibbonBar. When the application initiates the RibbonBar, it uses an argument in the wf\_init\_ribbonbar function to decide which way (XML or PowerScript) to use for create the RibbonBar (see [Initiate the RibbonBar](#_Initiate_the_RibbonBar)).

**Creating the XML for the RibbonBar in this demo (Recommended):**

A dedicated tool, RibbonBar Builder, is available in PowerBuilder to assist you to create the RibbonBar XML. In the tool, you can use a template (RibbonBar.xml) for creating your own XML, and preview the RibbonBar UI while you work. This is the recommended approach.

In the Sales Demo application:

* Open Tools > RibbonBar Builder in PowerBuilder IDE, and open and edit the file created for the RibbonBar: *SalesApplicationDemo\_RibbonBar.xml*.

**Creating the PowerScript for the RibbonBar in this demo:**

In the Sales Demo application:

* Open the function w\_mdi.wf\_init\_ribbonbar, and you will see the PowerScript for the RibbonBar in the function. The script is different from XML, but shall still be easy to understand.

For example, the following script inserts File as a RibbonBar category, add a orderview panel in this category, and then add buttons inside the panel.

*//File menu*

*long ll\_handle,ll\_category,ll\_panel,ll\_group*

*RibbonSmallButtonItem lrs\_item*

*Ribbonmenu lrm\_menu*

*ribbonLargeButtonItem lrl\_item*

*ribbongroupitem lrg\_item*

*long ll\_return,ll\_tmp,ll\_tmp2*

*ll\_category = arbb.insertcategoryfirst("File") //Insert a category*

*//orderview*

*ll\_panel = arbb.insertpanellast( ll\_category, "OrderView","orderview.png") //Insert a panel*

*arbb.insertLargebuttonlast(ll\_panel,"Orders","orderview.png","ue\_orderview") //Insert button*

*arbb.insertLargebuttonLast(ll\_panel,"Print","printbig.png","ue\_orderview\_print")*

# Add the RibbonBar to the application

## Replacing the existing menu with an empty one

After the RibbonBar is applied, the previous menu willno longer be used in the application. We shalluse an empty menu to replace the previous application menu assigned to the MDI window.

1. Use New > PB Object > Menu to create a new menu object. It is unnecessary to define any menu item for the menu or attach any script to it. If you do add a menu item, make sure the item shall be set to invisible.
2. Assign the empty menu to the MDI window.

In the Sales Demo application, we created the menu object m\_mdi\_none and assigned it to mdi\_1. The menu object contains one main menu and one submenu. Note that the submenu item is set to invisible.

## Insertinga RibbonBar control to the MDI window

Insert the RibbonBar control object into the MDI window. Note that you must resize the width and height for RibbonBar area in correspondence with the MDI client area to make sure the RibbonBar fits well in the MDI window.

In the Sales Demo application:

1. Insert the RibbonBar control into the MDI window mdi\_1
2. Addthe following resize() code toresize the RibbonBar and the MDI window:

*//resize RibbonBar*

*rbb\_1.move(0,newheight - this.workspaceheight() )*

*rbb\_1.width = newwidth*

*//resize mdi\_1 based on RibbonBar*

*mdi\_1.move(0,rbb\_1.height +newheight - this.workspaceheight())*

*mdi\_1.resize(newwidth,newheight - rbb\_1.height - ( newheight - this.workspaceheight()))*

## DisablingControlMenu insheet windows

Because the RibbonBar control applies to the whole application, it is no longer necessary to provide the Control Menu in sheet windows. To make sure the sheet window still provides the Close button after the Control Menu is removed, the new RibbonBar shall contain the Close/Exit button.

In the Sales Demo application:

1. Go to every sheet window (for example, w\_sheet), and uncheck the ControlMenu property in General property of the window.
2. Add the close/exit button in the RibbonBar top-right corner when we [Design and create a RibbonBar](#_Design_and_create).

# Initiate the RibbonBar

In the [Design and create a RibbonBar](#_Design_and_create) section, we used two ways to createtheRibbonBar for the application: by XML or PowerScript. We can choose to initiate the RibbonBar created in either way.

## Initiating the RibbonBar createdby XML

In the Sales Demo Application:

1. Add an instance variable in the MDI window.

*String is\_Ribbonbar\_XML\_Name = "SalesApplicationDemo\_RibbonBar.xml"*

1. Initiate RibbonBar menu in the MDI window open() event. The second argument is True, indicating that the RibbonBar will be initiated using XML:

*wf\_init\_ribbonbar(rbb\_1,True)*

1. In the function wf\_init\_ribbonbar of MDI window, load the RibbonBar from theXML file with the following code:

*If ab\_LoadXML Then*

*arbb.LoadFile(is\_Ribbonbar\_XML\_Name)*

*return*

*End If*

## Initiating the RibbonBar created by PowerScript

In the Sales Demo Application:

* Initiate RibbonBar menu in the MDI window open() event. The second argument is False, indicating that the RibbonBar will be initiated using PowerScript:

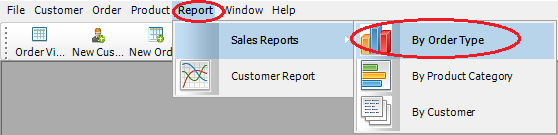
*wf\_init\_ribbonbar(rbb\_1,False)*

# Associate functions with RibbonBar items

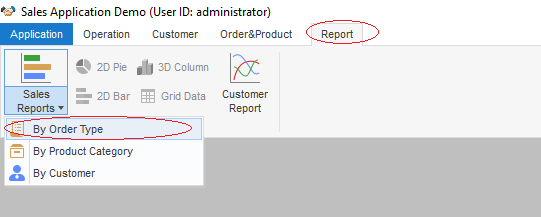
Previously the application functions are all associated with menu items. You need to decide which function to associate with which RibbinBar item now. In the Sales Demo application, you can see clearly how a function that is previously associated with a menu item is now associated with a RibbonBar item. In this tutorial, let’s explain two examples.

## Example 1– Opening the “By Order Type” report

In the original application, the “By Order Type” report is opened from Report > By Order Type menu item.



The new application uses the Report > Sales Reports > By Order Type RibbonBar item to open the report instead.



**PowerScriptinthe previous menu item m\_report1.m\_salesreports.m\_byordertype:**

*str\_rptparm lstr\_parm*

*lstr\_parm.ftitle = 'Sales Report by Order Type'*

*lstr\_parm.fdataobject = ""*

*opensheetWithParm(w\_rpt\_order\_type,lstr\_parm , parentwindow , 0 , Original! )*

**PowerScript inthe event ue\_rep\_ordertype of the RibbonBar:**

*str\_rptparm lstr\_parm*

*lstr\_parm.ftitle = 'Sales Report by Order Type'*

*lstr\_parm.fdataobject = ""*

*opensheetWithParm(w\_rpt\_order\_type,lstr\_parm , parent , 0 , Original! )*

*iw\_tmp = w\_rpt\_order\_type*

*wf\_setstyle("2D BarStacked")*

**Key points covered in the PowerScript change:**

* *opensheetWithParm(w\_rpt\_order\_type,lstr\_parm , parent , 0 , Original! )*

As the report item is placed in RibbonBar panel, the parent object for this item is changed. The mdiframe argument of OpenSheetWithParm is changed from parentwindow to parent. The Tag value is migrated into the this RibbonBar item. You can find more information for Tag migration in next example.

* *iw\_tmp = w\_rpt\_order\_type*

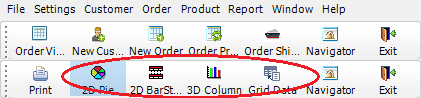
The original toolbar items are migrated as RibbonBar items (small buttons), e.g. 2D Pie, 3D Column, 2D BarStacked/Line and Grid Data items. The items may be enable or disable depending on the activation of Sheet window. Here we add each opened window into a buffer to track the activation status, so that it can control the status for the RibbonBar items.

* *wf\_setstyle("2D BarStacked")*

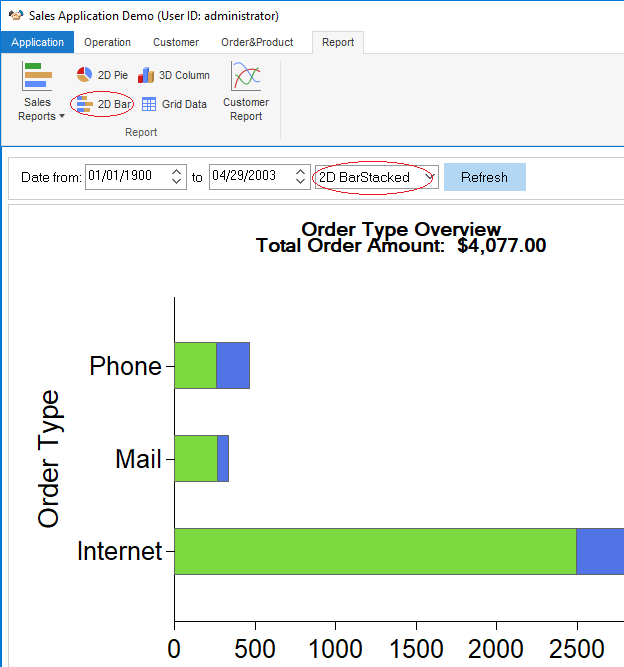
Identify if the report supports 2D BarStacked or 2D Line style. In this example, the small button “2D Bar” is displayed when this report is actively opened.

## Example 2–Opening the “2D BarStacked” report

In the original application, the “2D BarStacked” report is opened from the “2D BarStacked” toolbar item.



The new application uses the Report > 2D Bar RibbonBar item to open the “2D Bar” report, and then provides the 2D BarStacked option in the report for users to view the 2D Bar Stacked report.



**PowerScript in theprevious toolbar item m\_report.m\_settings.m\_defaultsettings.m\_reportstyle.m\_item1:**

*ParentWindow.Dynamic Event ue\_Settings(This.Tag)*

*If This.Checked Then Return*

*This.Checked = True*

*m\_item2.Checked = False*

*m\_item3.Checked = False*

*m\_item4.Checked = False*

*m\_item5.Checked = False*

**PowerScript in theevent ue\_report\_style of the RibbonBar:**

*ribbonsmallbuttonitem lrs\_item*

*this.getsmallbutton( al\_handle, lrs\_item)*

*string ls\_tag*

*ls\_tag = lrs\_item.tag*

*if isvalid (iw\_tmp) then*

*iw\_tmp.dynamic event ue\_Settings(lrs\_item.tag)*

*end if*

**Key points covered in the PowerScript change:**

* The Checked status for the toolbar item is no longer applicable for RibbonBar item.
* *ribbonsmallbuttonitem lrs\_item*

*this.getsmallbutton( al\_handle, lrs\_item)*

*string ls\_tag*

*ls\_tag = lrs\_item.tag*

The Tag property is used as identifier for some business logic of toolbar items. The Tag value need to be migrated into the RibbonBar item. In this example, we use handle to obtain the Tag value from a RibbonBar item. The report style will be switched based on the Tag value.

* *if isvalid (iw\_tmp) then*

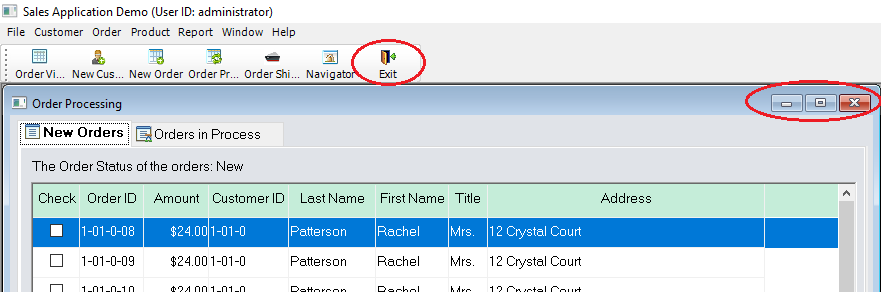
*iw\_tmp.dynamic event ue\_Settings(lrs\_item.tag)*

*end if*

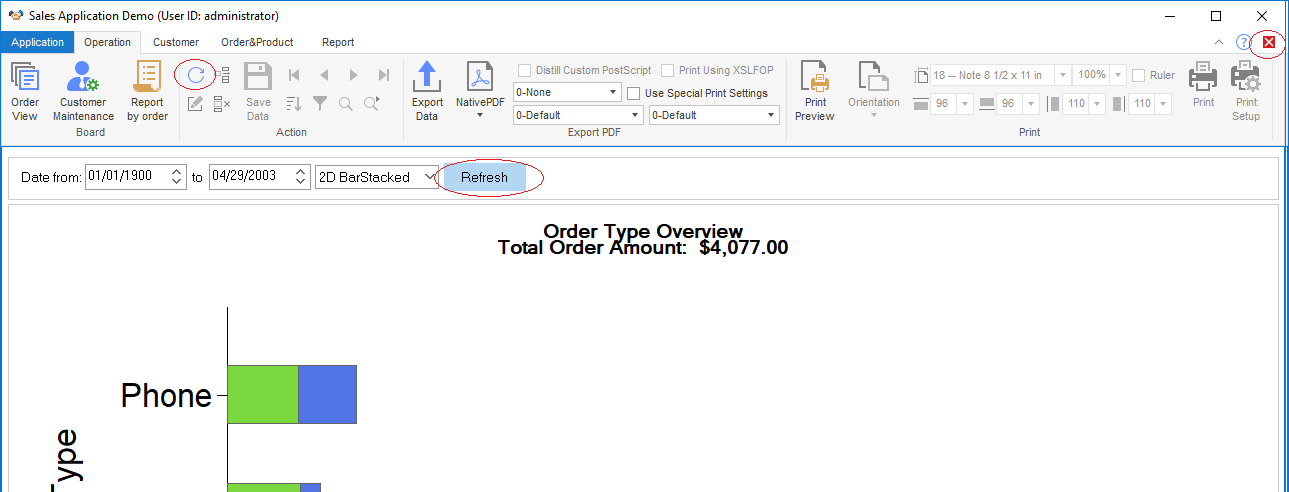
The ParentWindow is not applicable for RibbonBar items. However, the parent object is not applicable for RibbonBar either. Because the parent window is a MDI window instead of a sheet window. The actual parent object will be the tracked sheet window in the buffer. The buffer is maintained in the object iw\_tmp.

## Example 3 – Replacingthe Close/Exit insheet windows

In original MDI window application, the control menu provides the Close button to close the Sheet window, and the Exit button to close the application.



When we apply the RibbonBar, the ControlMenu property is disabledinthe sheet windows. The open, close and refresh functions are now provided in the RibbonBar instead.



**PowerScript inthe previous w\_sheet object close() event:**

*parentwindow().post dynamic event ue\_closesheet(this.classname())*

**PowerScript inthe w\_sheet object close() event of the RibbonBar:**

*parentwindow().post dynamic event ue\_closesheet(this.classname())*

*if isvalid(w\_mdi) then*

*w\_mdi.post function wf\_refresh\_ribbon()*

*end if*

*parentwindow().post function Arrangesheets(Layer!)*

**Key points covered in the PowerScript change:**

* *w\_mdi.post function wf\_refresh\_ribbon()*

In the Close event of the sheet window, call the wf\_refresh\_ribbon function to refresh the RibbonBar buttons and controls in the MDI window at the closing of the sheet window w\_mdi function. For example, if the 2D BarStacked report window is closed, the current active report supports2D Line style. The style button in the RibbonBaris refreshed to display the2D Line button.

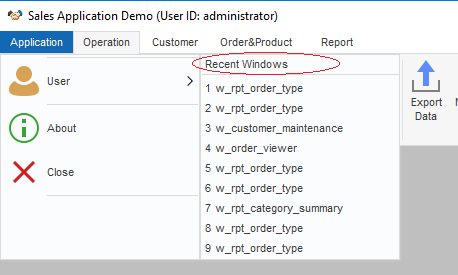
* *parentwindow().post function Arrangesheets(Layer!)*

In the Close event of the sheet window, call the ArrangeSheets function to re-arrange all the remaining sheet windows via the Layer parameter.

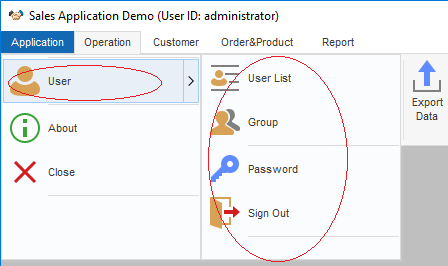
## Example 4 – Adding the Application Menu category

The Application Menu category contains the generic application functions, including:

* Recent Windows, listing the recently opened windows;



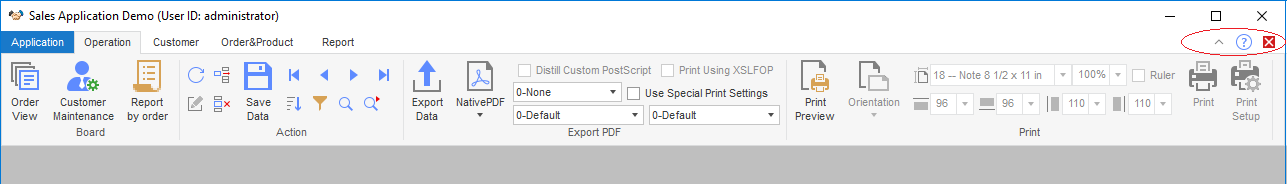
* User, providing the user/group management, password setting, and signout functions;



## Example 5 – Including special RibbonBar widgets

There are some special RibbonBar widgets that can further enhance user experience in the application.

For example, the standard RibbonBar buttons for collapsing and expanding, help, and closing functions, as shown below:



# Summary notes

* The RibbonBar can be constructed by a standard XML file. PowerBuilder provides RibbonBar Builder with preview functionality to assist you to create the XML file.
* When you apply a RibbonBar to an application, the previous menu in the application shall be removed. It is recommended that you create an empty menu and assign it to the MenuName property of the MDI window of the application.
* The ControlMenu property must be disabled for all windows in the application, so that the ControlMenu titlebar won’t display when a sheet window is maximized.
* Remember to resize the width and height for RibbonBar area and the MDI client area after the RibbonBar is added to the MDI window. This is to make sure the sheet windows opened in the MDI window will display properly together with the RibbonBar.
* If some windows are inherited from an ancient window, you can follow the same way to associate functions with the child windows as with the ancient window.

If previously some menu items with same functionality are placed in different sheet windows and you plan to put the menu items in the same panel in the RibbonBar, you can consider to control the enable/disable statuses of the RibbonBar items depending on the active status of the sheet windows.