Honeywell

Lab Exercise – Consolidate Code and Dynamically Position Dialog

57311006L
11/99

TotalPlant

Notices and Trademarks

This module supports **TotalPlant** Solution (TPS) system network.

TPS is the evolution of TDC 3000^{X}.

Honeywell Inc.
Industrial Automation and Control
Automation College
2820 West Kelton Lane
Phoenix, AZ 85053-3028
1-800 852-3211

Lab Exercise 6

Introduction

In this lab exercise you accomplish several objectives. By now, you should have a good understanding about the interaction of the target and popup dialog display parameters. As a result, the next two lab exercises combine several display enhancements to help alleviate the tedium of replacing embedded displays.

As you have probably noticed by now, you have quite a few scripts on your popup dialog with more to come. This points out an interesting challenge in any type of scripting language. Because scripting languages allow you to code freely on your objects, it can become a housekeeping or management problem of keeping track of where all your scripts reside. Because of this potential problem, you can take an approach where you try to place as much code as possible on one object. Taking this approach means that you can better document and manage your scripts. In this lab exercise, you will consolidate most of your popup dialog's code that you have entered in earlier exercises on the EXIT button.

Additionally code is added to make the dialog "popup" wherever it is needed in your display. This means the code makes the dialog visible and invisible, as well as position the dialog next to a process element (that is, at desired x and y pixel coordinates) in your GUS display.

Objectives

At the end of the lab exercise, you will be able to do the following:

- Consolidate display object scripts on the EXIT button using sub-procedures and functions that you create.
- Position the popup dialog and make it visible whenever the digital target is selected.
- Add EXIT button functionality to the dialog.

Design Criteria – Digital Target

The changes to the digital rectangle target are minor. In the lab exercise, all you will have to add are two display parameters that support the position of the popup dialog at x and y display coordinates:

- You will add two display parameters that define the x and y coordinates for the positioning of the popup dialog. The coordinates typically specify where the dialog appears, which is usually in an open area next to a process element.
- Modify the OnLButtonUp script that you had coded earlier in the digital target to pass the x and y coordinate for the popup dialog.

The display parameters that you will add to the digital target are the following:

- x – this parameter represents an x coordinate for the popup dialog.
- y – this parameter represents a y coordinate for the popup dialog.

Code that will be added to your digital target is shown in **bold**:

```
Sub OnLButtonUp() 'touchscreen and mouse click event
       Tag.external = pname      'send tagname string to dispdb.ent

           'send display.params to popup dialog
       display.params.obj.params.PName = pname
       display.params.obj.params.State0 = State0 'Button2
       display.params.obj.params.State1 = State1 'Button1
       display.params.obj.params.State2 = State2 'Button3
       display.params.obj.params.PtDesc = PtDesc
       display.params.obj.params.PntType = PntType
       display.params.obj.params.NoStates = NoStates
       display.params.obj.params.x = display.params.x
       display.params.obj.params.y = display.params.y
       display.params.obj.params.Info_File = display.params.Info

           'other targets check dispb.str01
       me.visible = true 'target is selected
       dispdb.str01 = My_name

End Sub
```

Design Criteria – Popup Dialog Code Consolidation

Consolidating your code makes it easier to manage your script. One programming technique is to place as much code as possible onto a single display object. For the popup dialog, you will move all of your script from the INFO and State change buttons and the alarm text object to the EXIT button's rectangle object. To do this means inserting your own *custom* sub procedures. Then, for example, when the operator selects a State change button, an OnLButtonUp event causes a call to your 'custom' sub-procedure to execute the related procedure.

For example, the state change button script for Button1 is currently coded as:

```
      'state1 button, button1
Sub OnLButtonUp()
       On Error Goto Button_error

       display.params.Tag.op = display.params.State1

       Exit Sub
       Button_error:
              INFO.fillcolor = tdc_red 'INFO button goes red
              MsgBox "Error '" & Err & " - " & Error$ & "'"
End Sub
```

Each state change button has similar code.

After consolidation, the code appears as the following on the Button 1 rectangle.

```
    'see EXIT button for the code
Sub OnLButtonUp()
      Call BUTTON1_LClick()
End Sub
```

After consolidation, the Button1 custom procedure code appears, along with your other scripts, as the following on the EXIT button (i.e., the EXIT rectangle object):

```
Sub BUTTON1_LClick()'this is your custom subroutine that is called
       On Error Goto Button_error

       display.params.Tag.op = display.params.State1

       Exit Sub
       Button_error:
              INFO.fillcolor = tdc_red 'INFO button goes red
              MsgBox "Error '" & Err & " - " & Error$ & "'"
End Sub
```

In other words, you create subprocedures that are called by LButtonUp events from other objects. Most of the custom subprocedures are located on one object (in this case, the EXIT button).

Just before the lab exercise there is an example that shows what your code should look like on the popup dialog after code consolidation. But first, review a description of some popup dialog functionality that you will need to add.

Design Criteria – Popup Dialog Positioning and Visibility

To reposition the popup dialog requires that you make use of the dialog's trans x and trans y properties. These properties can be best thought of as 'offsets from the object's origin or registration point.' The registration point represents the upper left corner x, y pixel position of the display object.

For example, if a display object is inserted in the upper left corner of a display, its registration point is 0, 0. All trans x and trans y values are then offset in pixels from 0, 0. In order to make the popup dialog position itself at a desired location, you need to send the x and y offsets from the digital target to the popup dialog.

To make the dialog visible or invisible requires grouping all objects that make up the popup dialog. You can group the dialog and then give the grouped objects a name, such as 'Pop.' Because the dialog is now grouped, you can make the dialog visible and invisible by referencing the group name, Pop, from script. The script, like most of your consolidated script, can reside on the EXIT button.

In addition, the dialog needs to be invisible at display startup or when the EXIT button is chosen. The popup dialog should also appear at the desired x, y position whenever a digital point target is selected. This will require scripting that creates a function to make the dialog visible (or invisible) and positioned. The function is passed a boolean value ("b") to make the dialog visible or invisible and x, y coordinates for positioning. The function is shown below:

```
      'popup dialog group (pop)set to x,y location
Sub Show(b as boolean)
       Pop.TransX = display.params.x
       Pop.TransY = display.params.y
       Pop.visible = b
       Pop.selectable = b
End Sub
```

An example procedure that calls the function to make the dialog invisible is shown below:

```
Sub OnDisplayStartup() ‘popup dialog set invisible at startup
       call show(false)
End Sub
```

The second challenge that must be met is to make the dialog visible when a digital point type is selected. To accomplish this, you can check with a flow control statement (such as If…Then) to see whether a digital point is selected. Recall from an earlier lab exercise that one of the display parameters that is defined for the popup dialog is PntType (point type). This information is passed to the popup dialog from the target. The code fragment that checks point type is shown below:

```
If (pnttype = "DEVCTL" or pnttype = "DIGCOM" or _
       pnttype = "DIGOUT") Then
       call Show(true) ‘position and make popup visible
Else
       call Show(false) ‘make popup invisible
       Exit Sub
End If
```

In order to accomplish dialog positioning, you will add two display parameters to the popup dialog that receive the x and y coordinates for the positioning of the popup dialog. The display parameters that you will add to the popup dialog are the following:

- x – this display parameter represents an x coordinate for the dialog.
- y – this display parameter represents a y coordinate for the dialog.

In summary, to update the popup dialog and consolidate the code means that you have to move and revise your code on your Alarm text object, INFO button, and the 3 State change buttons over to the Exit button. These objects now make calls to subprocedures on the Exit button.

When finished with the lab exercise, your consolidated code would appear as the following statements on the EXIT button. (Note: In your EmbedLab6 folder, a .txt text file is provided that contains code to save you typing time. From that file you can copy the fragments of code that you need for your lab exercise.)

```
Sub OnDisplayStartup()    'popup dialog set invisible at startup
       call show(false)
End Sub

'popup dialog group (pop)set to x,y location
Sub Show(b as boolean)
       Pop.TransX = display.params.x
       Pop.TransY = display.params.y
       Pop.visible = b
       Pop.selectable = b
End Sub

Sub OnDataChange()
       On Error Goto ODC_Error
       dim almstat as String 'point alarm status
       dim pnttype as string 'digital point type
       dim clr as long 'info button color

       pnttype = display.params.Pnttype

       If (pnttype = "DEVCTL" or pnttype = "DIGCOM" or _
              pnttype = "DIGOUT") Then
              call Show(true) 'position and make popup visible
       Else
              call Show(false) 'make popup invisible
              Exit Sub
       End If

       almstat = display.params.Alarm
       'check alarm status
       if almstat = "NOALARM" Then
              ALM.visible = false
              ALM.blink = false
       elseif almstat = "UNAKALRM" Then
              ALM.visible = true
              ALM.blink = true
       elseif almstat = "AKDALRM" Then
              ALM.blink = false
              ALM.visible = true
       end if

       clr = display.params.Info_color
       If clr = 0 Then
              INFO.fillcolor = makecolor(204,204,204)
       Else
              INFO.fillcolor = display.params.Info_color
       End If
Exit Sub
       ODC_Error:
End Sub
```

```
‘procedure for INFO button click
Sub INFO_LClick()
	On Error Goto Info_Error

	INFO.fillcolor = makecolor(204,204,204)
	If FileExists(display.params.Info_File) Then
		dispdb.ent01G.external = display.params.Tag.[name]
		WinSize 8500,8500
		InvokeDisplay(display.params.Info_File)
	Else
		detail display.params.Tag.[name]
	End If
Exit Sub

	Info_Error:
End Sub

‘procedure for State1 button click
Sub BUTTON1_LClick()
	On Error Goto Button_error

	display.params.Tag.op = display.params.State1

	Exit Sub
	Button_error:
		INFO.fillcolor = tdc_red
		MsgBox "Error '" & Err & " - " & Error$ & "'"
End Sub

‘procedure for State0 button click
Sub BUTTON2_LClick()
	On Error Goto Button_error

	display.params.Tag.op = display.params.State0

	Exit Sub
	Button_error:
		INFO.fillcolor = tdc_red
		MsgBox "Error '" & Err & " - " & Error$ & "'"
End Sub

‘procedure for State2 button click
Sub BUTTON3_LClick()
	On Error Goto Button_error

	display.params.Tag.op = display.params.State2

	Exit Sub
	Button_error:
		INFO.fillcolor = tdc_red
		MsgBox "Error '" & Err & " - " & Error$ & "'"
End Sub

‘procedure for EXIT button
Sub OnLButtonUp()
	INFO.fillcolor = makecolor(204,204,204)

	Show(false)

End Sub
```

Your code for the Alarm text object, ALM, should be deleted after you have copied it to the EXIT button's OnDataChange script.

Your code for the INFO button should be revised by deleting its OnDataChange script. The code that executes for the OnLButtonUp event will be "called" from the EXIT button. The OnLButtonUp script should appear as the following **boldfaced** statements:

```
Sub OnLButtonUp()
      Call Info_LClick() 'procedure on EXIT button
End Sub
```

Your code for the OnLButtonUp script on the State1 button (i.e., the Button1_State1 rectangle object) should appear as the following **boldfaced** statements.

```
Sub OnLButtonUp()
      Call Button1_LClick() 'procedure on EXIT button
End Sub
```

Your code for the OnLButtonUp script on the State0 button (i.e., the Button2_State0 rectangle object) should appear as the following **boldfaced** statements.

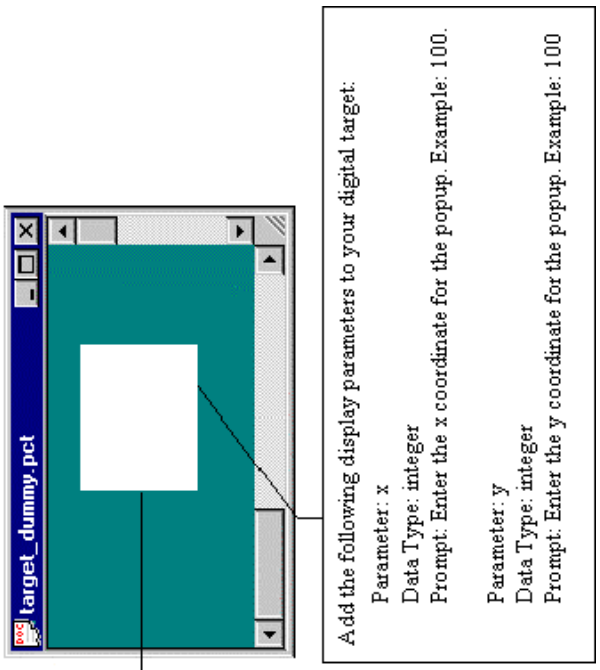
```
Sub OnLButtonUp()
      Call Button2_LClick() 'procedure on EXIT button
End Sub
```

Your code should appear as the following **boldfaced** statements on the State2 button (i.e., the Button3_State2 rectangle object).

```
Sub OnLButtonUp()
      Call Button3_LClick() 'procedure on EXIT button
End Sub
```

At the end of your lab exercise, there should be only one click event that contains the call procedure for each of the above-mentioned buttons. All other script should now be on the EXIT button.

The following figure summarizes the digital target design changes.



Code that is revised in your digital target is shown in **bold**:

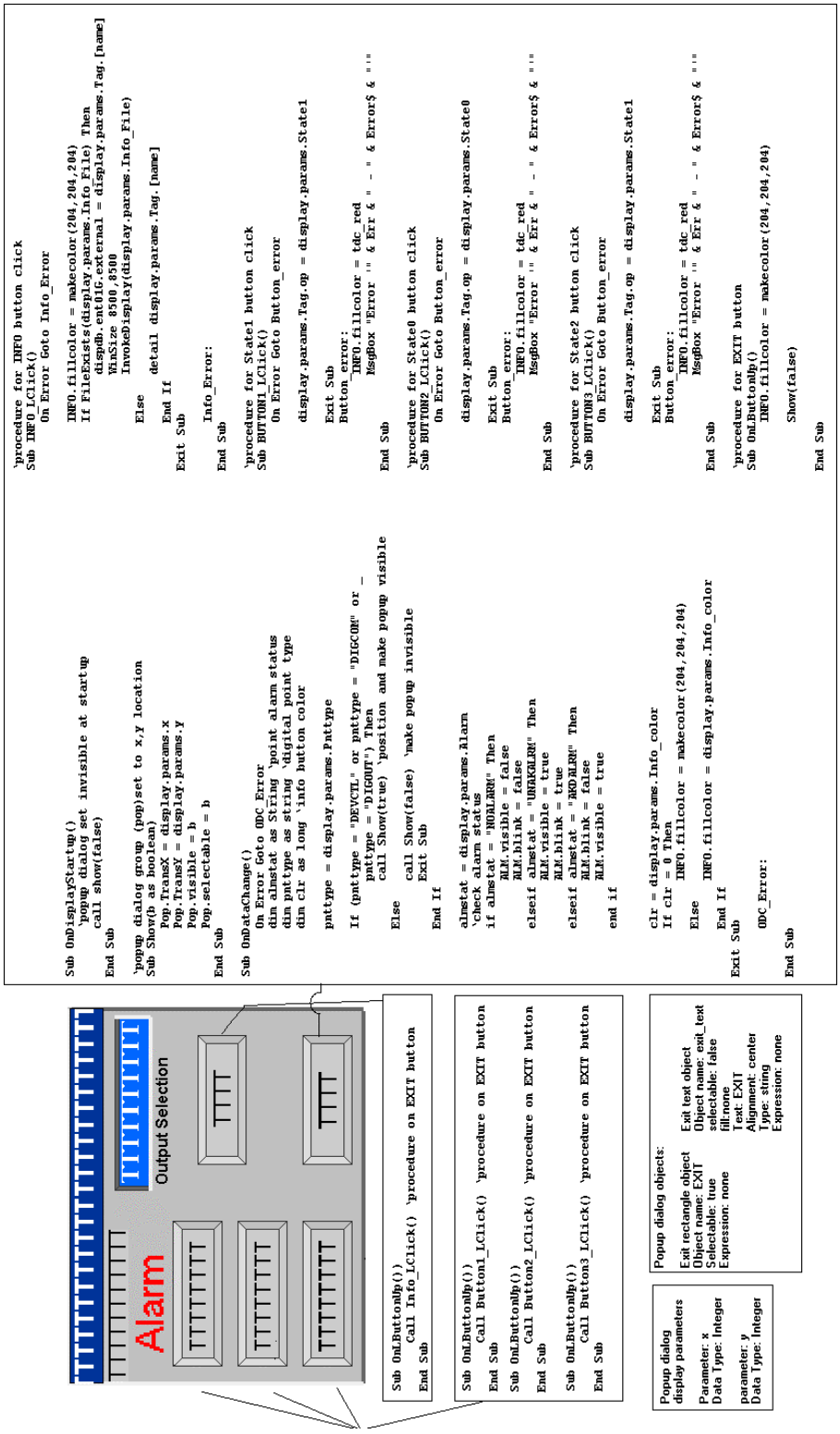
```
Sub OnLButtonUp()  'touchscreen and mouse click event
    Tag.external = pname  'send tagname string to dispdb.ent

    'send display.params to popup dialog
    display.params.obj.params.PName = pname
    display.params.obj.params.State0 = State0  'Button2
    display.params.obj.params.State1 = State1  'Button1
    display.params.obj.params.State2 = State2  'Button3
    display.params.obj.params.PtDesc = PtDesc
    display.params.obj.params.PntType = PntType
    display.params.obj.params.NoStates = NoStates
    display.params.obj.params.x = display.params.x
    display.params.obj.params.y = display.params.y
    display.params.obj.params.Info_File = display.params.Info

    'other targets check dispb.str01
    me.visible = true  'target is selected
    dispdb.str01 = My_name

End Sub
```

The following figure summarizes the popup design changes.



Lab Procedure – Modify the Digital Target to Send x and y coordinates

Step	Action
1.	From your EmbedLab5 folder, open the digital target display target5.pct.
2.	Save this display as target6.pct into your EmbedLab 6 folder.
3.	Add the following display parameters to your digital target: • Parameter: x • Data Type: integer • Prompt: Enter the x coordinate for the popup. Example: 100. • Parameter: y • Data Type: integer • Prompt: Enter the y coordinate for the popup. Example: 100
4.	Validate your target6.pct display. (Ignore any errors against inline data types.)
5.	Save this display as target6.pct into your Embed Lab 6 folder.

6.	Add script to this target's OnLButtonUp event so that the x and y coordinates can be transferred to the popup dialog. The script follows the design criteria scripting example on the previous pages; the code fragment that you will need to add is listed below in **bold**:
	`{rest of your code that you entered earlier }`
	
	`Sub OnLButtonUp() 'touchscreen and mouse event`
	
	`{rest of your code that you entered earlier }`
	`display.params.obj.params.PntType = PntType`
	`display.params.obj.params.NoStates = NoStates`
	`display.params.obj.params.x = display.params.x`
	`display.params.obj.params.y = display.params.y`
	`display.params.obj.params.Info_File = display.params.Info`
	
	`{rest of your code that you entered earlier }`
	`End Sub`

7.	Check your syntax. (Ignore any errors against inline data types.)
8.	Validate your target6.pct display. (Ignore any errors against inline data types.)
9.	Save this display as target6.pct into your Embed Lab 6 folder.

Lab Procedure – Modify the Popup Dialog

Step	Action
1.	From your EmbedLab5 folder, open the popup dialog display, dig_dialog5.pct, from the previous exercise.
2.	Save the display as dig_dialog6.pct into your Embed Lab6 folder.
3.	Add the following display parameters to your popup dialog to receive values from the target: • Parameter: x • Data Type: integer • Parameter: y • Data Type: integer (Note: No prompts are needed since the target passes the x, y values to the dialog.)
4.	Modify the EXIT text object and EXIT rectangle object so that it can support operator interactivity. Modify the EXIT text object so that it has the following properties: • Text object name: exit_text • Selectable: false • Fill: none • Text: EXIT • Alignment: center • Type: string • Expression: (no entry) Modify the EXIT rectangle object so that it has the following properties: • Rectangle object name: Exit • Selectable: true
5.	**Add the consolidated code as outlined in the Design Criteria section**. You can move the code from your existing • INFO button, • the 3 State change buttons, and • Alarm text object (Note: OR, you could copy code from a text file in your EmbedLab6 folder).

6.	Verify that the following OnLButtonUp script appears on the rectangle object used as the EXIT button.
	
	`'procedure for EXIT button`
	`Sub OnLButtonUp()`
	`  INFO.fillcolor = makecolor(204,204,204)`
	
	`  Show(false)`
	
	`End Sub`

7.	Select all of the objects in your popup dialog (Choose Edit>Select All).
8.	Group all of the objects in your popup dialog (Choose Draw>Group).
9.	Give the grouped objects a new name of **Pop**. (Note: Recall that your script makes the dialog visible and invisible by referencing the dialog as "Pop.")
10.	Validate the dig_dialog6.pct display.
11.	Save the display as dig_dialog6.pct in your Embed Lab6 folder.

Lab Procedure – Modify the Display

Step	Action
1.	From your EmbedLab5 folder, open your embed5.pct display from the previous lab exercise.
2.	Save this display as embed6.pct into your Embed Lab 6 folder.
3.	Replace your previous target5.pct(s) from the embed5 display and insert the new target6.pct (Choose Edit>Replace Embedded Display).
4.	Identify the x, y coordinates of where you want the popup to appear for each target. To do this, you can move the cursor to a desired location for the upper left-hand corner of the dialog and record the x, y coordinates that appear in the Status bar. Target1 x_____ target1 y_____ Target2 x_____ target2 y_____ (Example entries for the first target could be an x, y of 20 and 160, while entries for the second target could be an x, y of 330 and 160 if you are using embed.pct as your practice display.)
5.	Enter the x and y coordinates for both target6.pcts (Choose Edit>Enter Parameters.) • For the x coordinate, enter the x pixel position that you recorded above. • For the y coordinate, enter the y pixel position that you recorded above.
6.	Save this display as embed6.pct into your EmbedLab 6 folder.
7.	Replace the dig_dialog5.pct in the embed6 display and insert your new dialog, dig_dialog6.pct (Choose Edit>Replace Embedded Display). Move the digital dialog to the upper-left corner of your display (coordinates ~ 0,0).
8.	Validate your embed6.pct display.
9.	Save this display as embed6.pct into your EmbedLab 6 folder.
10.	Run the display. Expected result: The popup dialog should be invisible until you select a digital target.
11.	Select a digital target in your display. Expected result: The popup dialog appears next to the process element.
12.	Select another digital target in your display. Expected result: The popup dialog appears next to that process element.
13.	Choose your state change buttons to see if your custom sub procedures operate. Expected result: The desired state changes occur. (Note: To avoid a runtime error, your LCN point must be in MAN mode)
14.	Choose the INFO button. Expected result: The associated pct display or Native Window Detail display appears.
15.	Choose the EXIT button. Expected result: The popup dialog becomes invisible.

Last Page