Working with Screens

Table of Contents

> This section applies to non-Scene Graph based channels only. For best practices for Scene Graph channel program flow and navigation, see Controlling Screen Program Flow.

Screens

The primary user interface abstraction in the Roku object model is a 'screen'. There are a number of built-in types of screens for developers to use such as roSpringboardScreen and roPosterScreen. For example, the following code fragment will display a 'poster screen':

Example

```
port = CreateObject("roMessagePort")
poster = CreateObject("roPosterScreen")
poster.SetMessagePort(port)

' Set up contents of screen in a roAssociativeArray and pass to poster.SetContentList()

poster.Show()
```

Normally a screen is created by CreateObject(), then its contents are described, and finally it is displayed via Show(). Depending on the screen type, some of the contents may be described by explicit calls (such as SetTitle) and other contents are described by an array of *metadata* passed to SetContentList. The metadata format is described in Content Meta-Data.

The roAppManager object contains state that is common to all screens. You can have a common look and feel or apply themes to your app by setting attributes on the app manager.

Screen stack

The system internally maintains a stack of screens. Whenever a screen is created and displayed, it will be placed on the top of the screen stack. The top screen on the stack is the only one that is visible; all lower screens are hidden. All remote events are directed to the message port of the screen on top of the stack. When a screen is closed, it is removed from the stack and the screen which was below the top screen then becomes visible. If a screen closes and the screen stack is empty (that is, the only screen on the stack closed), then the app will immediately exit.

> When you create any of the standard screen types, you must call 'Show' in order for the screen to be displayed and added to the stack. However, roScreen (which supports 2D APIs) will be shown as soon as it is instantiated. Best practice is to create a screen when you are

ready to display it.

Normally a screen is closed when the screen object becomes unreferenced. Your code will have one or more variables which reference the screen object you created. When all such variables are destroyed (because their containing function has returned) or set to another value (such as 'invalid'), the screen becomes unreferenced. It is then closed and removed from the stack.

You must be careful when creating a new screen, not to let the other screens on the stack become accidentally unreferenced. For example, this could happen if you created both screens in one function and used the same variable for both screen objects.

Code structure

Generally, the best way to structure your code is to have a separate function for each screen. The function will create a new screen and a message port, set up and display the screen, and drive the event loop. If it wants to create a new screen, it will call the function for that screen from its event loop. This example shows a poster screen creating a springboard screen.

Example

```
Sub ShowPosterScreen()
 port = CreateObject("roMessagePort")
 poster = CreateObject("roPosterScreen")
 poster.SetMessagePort(port)
 ' Set up screen

 poster.show()
 While True
  message = Wait(0, port)
  If message.isScreenClosed()
   Exit While
  else if message.isListItemSelected()
   ShowSpringboardScreen( < get content details using > message.getIndex())
  end if
 End While
End Sub

Sub ShowSpringboardScreen()
 port = CreateObject("roMessagePort")
 springBoard = CreateObject("roSpringboardScreen")
 springBoard.SetMessagePort(port)
 ' Set up screen...

 springBoard.Show()
 While True
  message = wait(0, port)
  If message.isScreenClosed() Then
   Exit While
  Else If message.isButtonPressed() Then
   ' Process menu items...
  End If
 End While
    ' Returning destroys the 'springBoard' variable, which closes the
    ' springboard screen, and reveals the poster screen again.
End Sub
```

Non-stack model

Sometimes the stack model does not match your desired screen flow. For example, you may want to replace the screen on top of the stack with another one, rather than leaving the first screen on the stack. Most screens can be closed explicitly by calling the Close() method. This example shows a paragraph screen replacing another paragraph screen.

```
port = CreateObject("roMessagePort")
screen1 = CreateObject("roParagraphScreen")
screen1.SetMessagePort(port)
screen1.AddParagraph("first screen")
screen1.Show()

while true
    message = wait(0, port)
    if message.isScreenClosed() then
  exit while
 end if
end while

screen2 = CreateObject("roParagraphScreen")
screen2.SetMessagePort(port)
screen2.AddParagraph("second screen")
screen2.Show()
screen1.Close() ' close after new screen has been shown

while true
    message = wait(0, port)
    if message.isScreenClosed() then
  exit while
 end if
end while
```

This code displays the new screen **before** the old screen is closed. This is important both to avoid screen flicker and to avoid premature termination of your app. Recall that when the last screen on the stack is closed, the app exits. If you close the only screen on the stack and **then** try to create a new one, your app will exit before it gets a chance to create the new screen.

Facade screen

Another way to avoid having an empty screen stack is to create a *facade* screen which always remains on the stack until your app is ready to exit. An app that uses this technique might begin like this:

```
facade = CreateObject("roParagraphScreen")
facade.AddParagraph("please wait...")
facade.Show()
' leave facade on stack until app exits

... create other screens
```

The facade screen essentially keeps the app alive in case all other screens are closed.