SceneGraph Data Scoping

Table of Contents

SceneGraph applications have data object scoping rules that are similar to traditional programming languages. You have:

- *function scope*: objects that can only be accessed within the function in which they were defined
- *component scope*: objects that can be accessed anywhere within a component XML file, similar to *file scope* in traditional programming languages
- *global scope*: objects that can be accessed anywhere within a SceneGraph application

These different levels of scoping are identified by the use of the `m` object reference which can be used to disambiguate and access objects at different levels, similar to the use of `m` in BrightScript.

Function Scope

For creating objects with function scope, do not use the `m` object reference. For example, the following creates and defines several fields for a `dialog` object that can only be accessed within the function in which it is created and defined:

```
dialog = createObject("RoSGNode","Dialog")
dialog.backgroundUri = "pkg:/images/sgetdialogbg.9.png"
dialog.title = "Example Dialog"
dialog.optionsDialog = true
dialog.message = "Press * To Dismiss"
```

Component Scope

For creating objects with component scope, use the `m` object reference to identify objects that can be accessed anywhere within a component XML file. For example, if you wanted to create the same `dialog` object above in one function, but define it, or otherwise access it, in another function in the same component XML file:

```
sub createdialog()
  m.dialog = createObject("RoSGNode","Dialog")
end sub

sub definedialog()
  m.dialog.backgroundUri = "pkg:/images/sgetdialogbg.9.png"
  m.dialog.title = "Example Dialog"
  m.dialog.optionsDialog = true
  m.dialog.message = "Press * To Dismiss"
end sub
```

m.top Component Scope Reference

There is a special use of the `m` object reference in SceneGraph for identifying the *top* of the SceneGraph tree for a component XML file. To do this, use the `m.top` object reference, which refers to the component itself. For example, to create the `dialog` object and assign it to the `dialog` field of

a **Scene** node defined in a component XML file (which is the required usage of the **Dialog** node class), assign the `dialog` object to the `dialog` fie ld using the `m.top` object reference:

```
dialog = createObject("RoSGNode","Dialog")
dialog.backgroundUri = "pkg:/images/sgetdialogbg.9.png"
dialog.title = "Example Dialog"
dialog.optionsDialog = true
dialog.message = "Press * To Dismiss"
m.top.dialog = dialog
```

Likewise, if you want to use `findNode()` to find a SceneGraph node object anywhere in the SceneGraph tree for a component XML file, use the `m .top` reference to start at the top of the tree:

```
m.categorieslist = m.top.findNode("categorieslist")
```

Global Scope

Rules

- To declare a **data object** at **global scope**, store it in a field or child of the global node. This global node may be accessed from the entire SceneGraph application.
- To access the **global node** in **components**, use the predefined `m.global`, much like `m.top`.
- For access to the **global node** from **non-component script** as in `source/main.brs`, use **`getGlobalNode()`** called on the **`roSGScree n`** object.
- In **non-component script**, where the global node is obtained using **`getGlobalNode()`**, you can store it in **`m.global`** so that the syntax for subsequent references to it matches that for components.

For example:

```
screen = CreateObject("roSGScreen")
m.port = CreateObject("roMessagePort")
screen.setMessagePort(m.port)
m.global = screen.getGlobalNode()
m.global.id = "GlobalNode"
```

As noted, this is not necessary in component script, as `m.global` is predefined.

You can access and set the fields, or the children nodes, of the global node from anywhere in the SceneGraph application. In the non-component example above, the global node **`id`field** value is set to **`GlobalNode`**. Likewise, you can access and set the fields for the global node from components by accessing the component **`m`** to get its special global element:

> Note that you cannot edit elements within **associative arrays**. You will need to take the associative array, modify it and save it back into the field.

```
m.global.addFields( {red: &hff0000ff, green: &h00ff00ff, blue: &h0000ffff} )
...
m.rect = m.top.findNode("Rect1")
m.rect.color = m.global.red
...
color = m.rect.color
if m.rect.color = m.global.red
  m.rect.color = m.global.green
else if m.rect.color = m.global.green
  m.rect.color = m.global.blue
else
  m.rect.color = m.global.red
end if
```