Downloading Server Content

Table of Contents

Various types of content material can be downloaded from your server to a SceneGraph application. Many of the renderable nodes include a `uri` field that allows you specify the URI of a graphical image file or other content on your server. Setting this field causes the node to automatically download the content to the SceneGraph node, and for graphical images, be rendered on the display screen according the configuration of the node.

For many nodes that require more complex data, a `content` field is provided. This special field is designed to be assigned the value of a **Content Node** node that you configure to contain the data required for the node.

Configuring a ContentNode Node for Static Content

Generally, you'll want to download dynamic content for your application from your server. But to simply illustrate how to set up a **ContentNode** node for a node requiring data, the following shows setting a **ContentNode** node as a child of a **LabelList** node requiring the data using the `role` attribute. The example makes clear the hierarchal relationship required for the `content` field of a **LabelList** node, with a parent **ContentNode** node and several child **ContentNode** nodes for the list item text. The `role` attribute automatically assigns the **ContentNode** node to the `content` field of the **LabelList** node, and the SceneGraph application shows the text strings in the child **ContentNode** nodes as the items of the list.

Static ContentNode Node Configuration

```
<LabelList
  id = "moviemenu"
  translation = "[160,92]"
  itemSize = "[440,48]" >

  <ContentNode id = "moviemenucontent" role = "content" >

    <ContentNode title = "Comedy" />
    <ContentNode title = "Drama" />
    <ContentNode title = "Action" />
    <ContentNode title = "Horror" />

  </ContentNode>

</LabelList>
```

Configuring a ContentNode Node with Dynamic Content

The **ContentNode** node can be configured in a **Task** node when you want to download dynamic data from your server to the **ContentNode** node. To do this, you must write BrightScript code in the **Task** node to create the parent **ContentNode** and the hierarchy of child **ContentNode** nodes required for your application.

In a **Task** node, you can use the BrightScript **roUrlTransfer** component to read the content data from your server. If the data is contained in JSON or XML formats, you can then use the corresponding BrightScript parsing functions and components to configure a **ContentNode** node with the content data. The BrightScript JSON/XML parsing loop should be set up to terminate when the **Task** node **<interface>** field for the target **ContentNode** node is fully configured with new data, to allow an `observeField()` function set in the component that requires the data to trigger a callback function to assign the data to the target node.

For example, the following **Task** node downloads a list of strings to be used as text for a **LabelList** node. The **Task** node includes two **<interface>** fields, one for the URI of the content data (`uri`), and another for the **ContentNode** node that will be configured with the data (`content`). When an XML component requires a list of strings for a **LabelList** node, the component can set the `control` field of the **Task** node object to `RUN`. This sends the specified **<interface>** field values to the **Task** node, in this case the URI of the content data, and starts the **Task** node. The **Task** node then downloads an XML file with the content data from the specified URI, parses the XML data, and creates a parent **ContentNode** node, and child **ContentNode** nodes for each string. When the **ContentNode** node is configured with all the strings in the XML file, it is assigned to the **ContentNode** object reference set in the **<interface>** field. This change in the field value triggers the `observeField()` callback function in the component XML file to add the strings to the **LabelList** node `content` field.

First, here's the XML file the example **Task** node downloads from the server and parses:

Example Server XML File

```
<?xml version = "1.0" encoding = "UTF-8" standalone="yes" ?>

<listcontent>

  <item text = "Comedy" />
  <item text = "Drama" />
  <item text = "Action" />
  <item text = "Horror" />

</listcontent>
```

And here is how the component XML file configures and starts the **Task** node process, including setting the `observeField()` function to trigger a callback function to display the **LabelList** node after the content data is downloaded:

Example Task Node Configuration and Launch

```
m.getHomeOptionsList = createObject("roSGNode", "getLabelListContent")
m.getHomeOptionsList.setField("uri",
"http://www.sdktestinglab.com/homeoptionslistcontent.xml")
m.getHomeOptionsList.observeField("content", "showhomeoptionslist")
m.getHomeOptionsList.functionName = "showhomeoptionslist"
m.getHomeOptionsList.control = "RUN"
```

And here's the actual **Task** node itself that does the work of downloading the server XML file, parsing the XML content data, and configuring a **ContentNode** node with the data:

Downloading LabelList Node Content

```
<?xml version = "1.0" encoding = "utf-8" ?>

<!--********** Copyright 2015 Roku Corp.  All Rights Reserved. **********-->

<component name = "getLabelListContent" extends = "Task" >

<interface>
  <field id = "uri" type = "uri" />
  <field id = "content" type = "node" />
</interface>

<script type = "text/brightscript" >

  <![CDATA[

  sub init()
    m.top.functionName = "getContent"
  end sub

  sub getContent()
    content = createObject("roSGNode", "ContentNode")
    contentxml = createObject("roXMLElement")

    readInternet = createObject("roUrlTransfer")
    readInternet.setUrl(m.top.uri)
    contentxml.parse(readInternet.GetToString())

    if contentxml.getName() = "listcontent"
      for each item in contentxml.GetNamedElements("item")
        attributes = item.getAttributes()
        item = {
          text: attributes.text
          }
        listitem = content.createChild("ContentNode")
        listitem.title = item.text
      end for
    end if

    m.top.content = content
  end sub

  ]]>

  </script>

</component>
```

Note that this example could be used to download strings for any **LabelList** node, or for that matter, any node that requires a simple list of strings

as the required content for the node. For more complex data requirements, the XML parsing loop in the example could be expanded to include any **Content Meta-Data** attributes required for the node (and of course, contained in the server XML file).