ContentNode

Table of Contents

Extends: Node

Description

The **ContentNode** node class allows you to specify the data used to configure a node or component. Many nodes and components require a **ContentNode** node as the specification of their `content` field in order to be properly configured. In general, lists, grids, and panels require a **ContentNode** node for configuration. The data included in a **ContentNode** node can be data such as the text for labels in the node or component, and the spacing between items in a list, grid, or panel, including data to create custom lists, grids, and panels. The reference information for every node or component that requires a **ContentNode** node includes a section that details the requirements of the **ContentNode** node for that node or component.

ContentNode nodes defined as the specification for a node or component `content` field are typically structured as one **ContentNode** parent node, with a hierarchy of child nodes that specify the actual data, and sections of data if needed. For example, a **LabelList** node can have several sections that divide the entire list, each with their own section heading, and specific items in that section of the list. The **ContentNode** node for that **LabelList** node should have two levels of child **ContentNode** nodes, one level for the data to configure the list sections, and then another level of child **ContentNode** nodes for the data for each item in that list section.

A **ContentNode** node can also be used to specify the data for custom components with defined interfaces, and for nodes and components that require **Content Meta-Data**. Also, you should use a **ContentNode** node for complex structures of data for your application rather than associative arrays. **ContentNode** node objects are passed by reference in the application, while associative array objects are copied. For large complex data structures, passing **ContentNode** node objects is much quicker than passing the equivalent associative array object. You can use associative arrays for simpler data structures with just a few fixed members.

> All of the attributes listed in **Content Meta-Data** can be set as fields in a **Content** node. However, when creating a **Content** node, the fields themselves are not created until the valid attributes are set as fields, using either assignment (=), or set using **setField()** or **setFields()**.

Creating ContentNode Nodes

To create a **ContentNode** node object and populate it with data, you can define the **ContentNode** node in XML markup, or create it using BrightScript. In both cases, you will usually be first creating or defining a parent **ContentNode** node, then creating or defining children **ContentNode** nodes below the parent, with possibly other levels of children **ContentNode** nodes.

As an example, to define a **ContentNode** node with one level of children **ContentNode** nodes, you should generally:

```
<ContentNode role= or id= >
  <ContentNode content_meta-data_attribute = "attribute" ... />
  ...
</ContentNode>
```

The parent **ContentNode** node is defined with either an XML `role` attribute or an `id` field, depending on how you want to assign or use the content data. For the nodes classes that have a `content` field that is to be assigned a **ContentNode** node, such as **LabelList** nodes, the parent **ContentNode** node should be defined as a child node of the node:

```
<LabelList  id = "labellist" >
  <ContentNode  role = "content" >
    <ContentNode title = "Renderable Nodes" description = "Basic Nodes That Show Things" />
    <ContentNode title = "Z-Order/Parent-Child" description = "SceneGraph Tree Order Matters" />
    <ContentNode title = "Animations" description = "Moving Stuff Around and Flashing Lights" />
    <ContentNode title = "Events and Observers" description = "Reacting When Stuff Happens" />
  </ContentNode>
</LabelList>
```

In other cases, you can just use the `id` field to allow you to access the **ContentNode** node later as an object for any purpose. In both cases, define additional **ContentNode** nodes with content fields as children of the parent **ContentNode** node.

To create a **ContentNode** node in BrightScript, you should generally:

1. Create the **ContentNode** node object using `createObject()`
2. Create child **ContentNode** node objects using `createChild()`
3. Assign the data to the field(s) of each **ContentNode** child object

As follows:

```
ContentNode_object = createObject("RoSGNode","ContentNode")
ContentNode_child_object = ContentNode_object.createChild("ContentNode")
ContentNode_child_object.field_name = data
...
```

Assigning ContentNode Node Data To Nodes/Components

For nodes and components that require a **ContentNode** node as the specification of their `content` field, you can define it as a child of the node or component in XML markup using the `role` attribute, or just assign the **ContentNode** node object to the `content` field as follows:

```
NodeComponent.content = ContentNode_object
```

For other nodes and components that don't require a **ContentNode** node, you can use `getChild()` or a similar function to locate the specific child **ContentNode** node object that contains the data you want to assign to a particular node/component field:

```
ContentNode_child_object = ContentNode_object.getChild(child_number)
NodeComponent.field_name = ContentNode_child_object.field_name
```

> If you are assigning a component node field with child elements of a **ContentNode** node, you should create the **ContentNode** node and all child elements, then assign the **ContentNode** node to the component node field. It is more efficient than creating a **ContentNode** nod e, assigning the **ContentNode** node to the component node field, then creating and assigning each child element using **`getChild()`**. This is particularly true when creating and assigning **ContentNode** node data in different threads, which can be very inefficient. Also avoid getting individual data items or child elements repeatedly if it is more efficient to copy the data value locally and avoid the repetitive object accesses, particularly in two different threads.

Fields

All of the attributes listed in **Content Meta-Data** are accessible as fields using dot (.) notation on a **ContentNode** node object. For example, for a **C ontentNode** node object `iteminfo`, the **Content Meta-Data** `Description` attribute can be read or written as follows:

```
iteminfo.description
```

You can also access **ContentNode** attributes as fields using dot (.) notation if you add the attribute as an **<interface>** element field to an extended **ContentNode** component. For example, you could extend a **ContentNode** as a custom `listitemcontent` component with a `componentname` fi eld to include an XML component name in a list item:

```
<component  name = "listitemcontent" extends = "ContentNode" >
```

```
  <interface >
    <field id = "componentname" type = "string" />
  </interface>

</component>
```

Then for a `listitemcontent` **ContentNode** node object `iteminfo`, you can read or write the `componentname` field in the same way as if it were a **Content Meta-Data** attribute:

```
iteminfo.componentname
```

> You can *only* use dot (.) notation to access **ContentNode** fields as attributes found in **Content Meta-Data**, or defined as an **<interface>** element field in a custom **ContentNode** component.

Example

The following creates a component with a **LabelList** node populated with some specific content. To configure the content, a **ContentNode** node is created for the `content` field of the **LabelList** node. The **LabelList** node is divided into several sections, so for each section, a child **ContentNode** node object is added to the parent **ContentNode** node using the `addSection()` function. Then the individual items in each section of the list are added as child nodes of the section **ContentNode** node object using the `addItem()` function. The functions access the global variables for the **ContentNode** object reference `m.content` for the parent **ContentNode** node, and `m.sectionContent` for the section **ContentNode** nodes.

ContentNode Node Class Example

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

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

<component name = "NodeSelectionList" extends = "Group" initialFocus = "coreList" >

  <script type="text/brightscript" >

    <![CDATA[

    sub init()
      m.list = m.top.FindNode("coreList")

      m.content = createObject("RoSGNode","ContentNode")

      addSection("Renderable Nodes")
      addItem("Rectangle")
      addItem("Rotated Rectangle")
      addItem("Label")
      addItem("Poster")
      addItem("Video")
      addItem("Video Zoom")

      addSection("Animation Nodes")
```

```
      addItem("Animation Vector 2D Interpolator")
      addItem("Animation Color Interpolator")
      addItem("Animation Float Interpolator")
      addItem("Sequential Animation")
      addItem("Parallel Animation")
      addItem("Fade-In Animation")
      addItem("Fade-Out Animation")

      addSection("Control Nodes")
      addItem("Timer")

      addSection("Lists and Grids")
      addItem("Poster Grid")
      addItem("Markup Grid")

      m.list.content = m.content

      m.top.setFocus(true)
    end sub

    sub addSection(sectiontext as string)
      m.sectionContent = m.content.createChild("ContentNode")
      m.sectionContent.CONTENTTYPE = "SECTION"
      m.sectionContent.TITLE = sectiontext
    end sub

    sub addItem(itemtext as string)
      item = m.sectionContent.createChild("ContentNode")
      item.title = itemtext
    end sub

    ]]>

  </script>

  <children>

    <LabelList
      id = "coreList"
      translation = "[ 160, 92 ]"
      itemSize = "[ 440, 48 ]"
      itemSpacing = "[ 0, 0 ]"
      sectionDividerHeight = "48.0"
      sectionDividerFont = "font:MediumBoldSystemFont"
      sectionDividerTextColor = "0x880088FF" />
```

```
  </children>

</component>
```