PosterGrid Markup

Example Application: PosterGridExample.zip
Node Class Reference: **PosterGrid**

The **PosterGrid** node class shows a grid of graphic images that allow a user to select a content item or option from the grid.

The `postergridscene.xml` component file in `PosterGridExample.zip` has the following markup:

PosterGrid Example Expand source

```
<component name = "PosterGridExample" extends = "Scene" initialFocus =
"examplePosterGrid" >

  <script type = "text/brightscript" >

    <![CDATA[

    sub init()
      m.top.backgroundURI = "pkg:/images/rsgde_bg_hd.jpg"

      m.top.setFocus(true)

      m.postergrid = m.top.findNode("examplePosterGrid")

      m.postergrid.translation = [ 130, 160 ]

      m.readPosterGridTask = createObject("roSGNode", "ContentReader")
      m.readPosterGridTask.contenturi =
"http://www.sdktestinglab.com/Tutorial/content/rendergridps.xml"
      m.readPosterGridTask.observeField("content", "showpostergrid")
      m.readPosterGridTask.control = "RUN"
    end sub

    sub showpostergrid()
      m.postergrid.content = m.readPosterGridTask.content
    end sub

    ]]>

  </script>

  <children>

    <PosterGrid
      id = "examplePosterGrid"
      basePosterSize = "[ 512, 288 ]"
      caption1NumLines = "1"
      numColumns = "2"
      numRows = "2"
      itemSpacing = "[ 20, 20 ]" />

  </children>

</component>
```

The idea here is to define the basic dimensions and configuration of the **PosterGrid** node, the size of the graphic images, whether the image will have descriptive label, and the number of columns in the grid, then assign a **ContentNode** node with the grid content meta-data to the content field of the node to show the specific images and caption labels for a grid.

So we start by defining the basic dimensions and configuration of our example **PosterGrid** node in the **<children>** element of `postergridscene.xml`:

```
<children>
  <PosterGrid
    id = "examplePosterGrid"
    basePosterSize = "[ 512, 288 ]"
    caption1NumLines = "1"
    numColumns = "2"
    numRows = "2"
    itemSpacing = "[ 20, 20 ]" />
</children>
```

The `basePosterSize` field value should be set to the size of the graphic images in the grid. As with a **Poster** node, the actual downloaded image sizes may be different, and will be scaled to the size specified by the `basePosterSize` field value. Again, as with a **Poster** node, for the absolute best appearance of your graphic images in the grid, the downloaded image sizes should exactly match the `basePosterSize` field value.

You can control the appearance of any caption labels you add to the graphic images in your grid content meta-data, by setting fields such as `caption1NumLines`. In this example we have one caption label, which we limit to one line.

Then we further define the dimensions of the grid, specifically by setting the `numColumns` and `itemSpacing` field values. This controls the layout of the grid by limiting the number of columns in the grid to the value specified in the `numColumns` field, and each graphic image will be spaced from adjacent graphic images by the number of pixels specified by the `itemSpacing` field value (the array defines separate horizontal/vertical spacing values). The `numRows` field value will always be overridden by the actual number of images set in the content meta-data for the grid, but can be set to some anticipated average number of rows for maximum rendering efficiency.

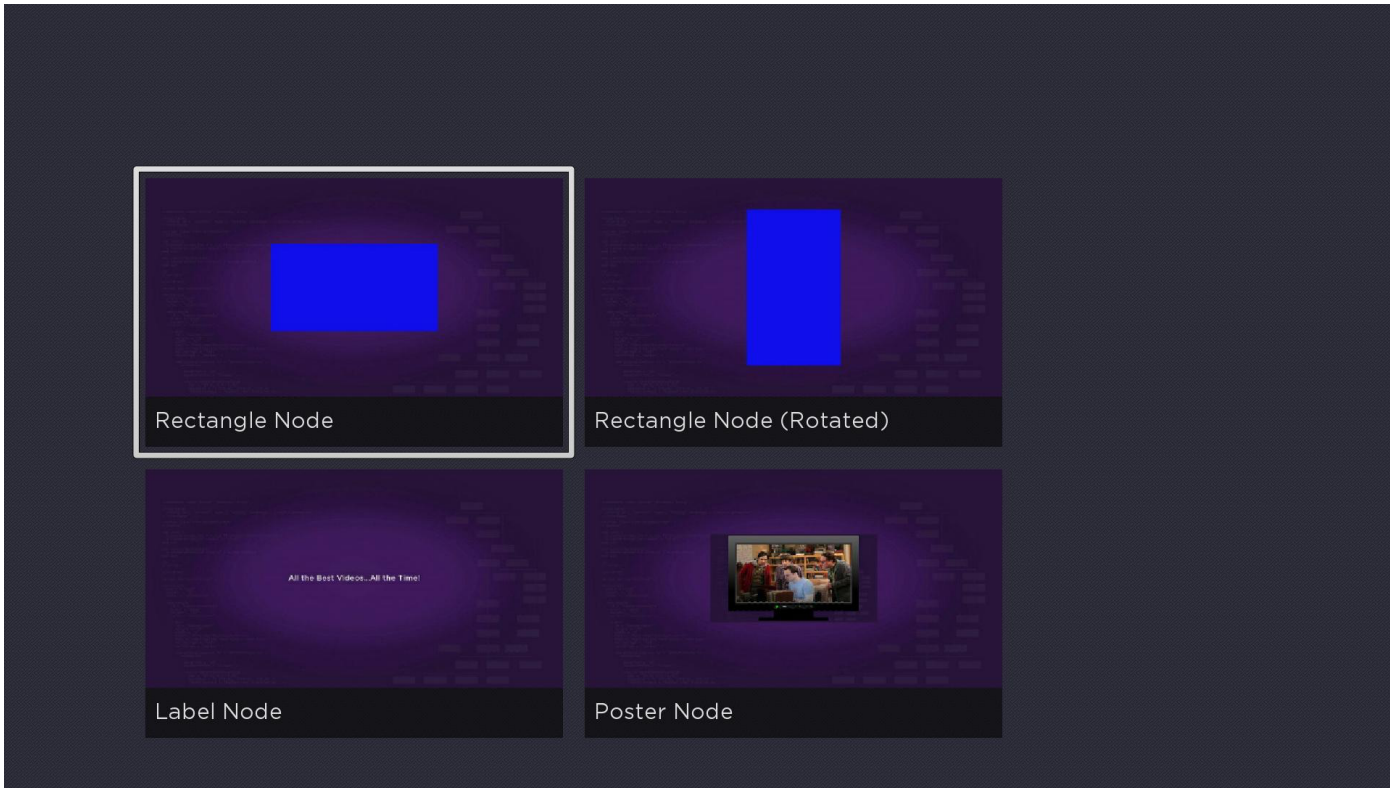
After defining the basic dimensions and configuration of the grid, the actual graphic images for the grid will be set in the **ContentNode** node assigned to the `content` field of the grid. In this example, we use the same content reader **Task** node seen in **Task Markup** to download and convert a server XML file to a **ContentNode** node. We create the content reader task object in the **<script>** element **`init()`** function, assign the **Task** node `contenturi` **<interface>** field with the URL of the grid content meta-data XML file, set an observer on the **Task** node **<interface>** `content` field, and set the **Task** node control field to `"RUN"` to start the asynchronous content reading task:

```
m.readPosterGridTask = createObject("roSGNode", "ContentReader")
m.readPosterGridTask.contenturi =
"http://www.sdktestinglab.com/Tutorial/content/rendergridps.xml"
m.readPosterGridTask.observeField("content", "showpostergrid")
m.readPosterGridTask.control = "RUN"
```

As soon as the **ContentNode** node is completed by the **Task** node, the change in the **Task** node `content` field triggers the `showpostergrid()` callback function which assigns the **ContentNode** node to the grid `content` field:

```
sub showpostergrid()
  m.postergrid.content = m.readPosterGridTask.content
end sub
```

And the result is:



You might want to look at the grid content meta-data XML file on the server:

http://www.sdktestinglab.com/Tutorial/content/rendergridps.xml

Note how the content meta-data attributes for each **PosterGrid** node item are set according to the attribute descriptions listed in **Content Meta-Data** and **PosterGrid**:

```
<Content >
  <item
    hdgridposterurl = "http://www.sdktestinglab.com/Tutorial/images/rectanglepg.jpg"
    shortdescriptionline1 = "Rectangle Node"
    x = "0" y = "0" />
  ...
</Content>
```