Playing Videos

Table of Contents

Playing any type of video requires just one SceneGraph node class: **Video**.

To play a video, you must first prepare the files to be served to the Roku Player:

- the video files must have been encoded in one of the formats supported by a Roku Player (see **Supported Streaming Formats**)
- if you want to use an adaptive bitrate switching scheme, the video files must be configured to use the supported adaptive bitrate scheme you want (see **Adaptive Bitrate Formats**)
- you must also include any supporting files for special playback options you want, such as *trick play* (see **Trick Mode Support**)

Then, in your application, you must:

- set the various playback configuration options for your video in a **ContentNode** node using **Content Meta-Data** (see **Content Meta-Data**) attributes
- assign the **ContentNode** node to the `content` field of the **Video** node
- set other **Video** node fields for all other video playback appearance and function configuration you want for your particular video (or videos)

Then, when a user selects a video to play, you set the **Video** node `control` field to `play`, in a callback function triggered by the user selection event.

Examples

Here are two example applications showing selecting and playing videos. These examples demonstrate the use of the **RowList** node class to create a custom content grid and **Video** node for video playback. The second example also shows how to create a very simple details screen for displaying more information about selected content before video playback. They also show how to load and parse an MRSS feed so you can see how to populate your channels with dynamic content.

- Simple_Grid_and_Video.zip
- Simple_Grid_with_Details_and_Video.zip

Here is another example application that uses the same content feed, but renders the UI as a PanelSet.

- PanelSet_and_Video.zip

And here is a simple example using the same basic content with a **LabelList** node for selecting the video to play. The XML file containing the **Cont**

ent Meta-Data is included and read from the `pkg:/server/videocontent.xml` file. Since the entire application including the **Content Meta-Data** is contained within the application package, you can see how the **Content Meta-Data** is parsed and translated into a **ContentNode** nod e, which supplies the items displayed in the **LabelList** node, and then how the video is played when the user selects a video from the list.

- VideoList.zip

Setting Focus on the Video Node

You must set focus on the **Video** node while playback is occurring for the trick play bar and some other user interface options to be used. After playback has finished, you will generally want to set focus back on the user interface element (usually a list or grid or custom video "details" screen) from which the user selected the video.

Setting Up Your Server Video Content Meta-Data Files

For your server content meta-data XML files, always ensure that all attribute strings containing the ampersand (&) character is properly escaped by replacing it with the HTML entity `&`. (This is particularly important because many video stream URLs include this character.) Also remember to properly escape all XML special characters in the attribute strings, and use the correct HTML entities to format your descriptive content meta-data attributes.

Media Playlists

Available since firmware version 7.2

Both the **Audio** and **Video** nodes support the use of playlists, which are lists of several media items to play in sequence rather than a single media item. If a playlist is used, setting the `loop` field to true causes the entire playlist to play again after the last item in the playlist completes.

To set an **Audio** or **Video** node to play several media items in sequence:

1. Set the `content` field to a **ContentNode** node containing a child **ContentNode** node for each media item in the playlist.
2. Set the `contentIsPlaylist` field to true.

After setting the `control` field to `play`, you can control the playback as follows:

- to stop playing the current media item in the playlist, or end playback if the current item is the last in the playlist, set the `skipcontent` opti on of the `control` field
- set the `nextContentIndex` field to an index of another item in the playlist to be played after the current item finishes, rather than the next item in the sequence
- for **Audio** nodes, you can observe and use the `contentIndex` field value to control the appearance of the screen while the audio item is playing, such as showing the title of a song when that song begins playing

Configuring Video Stream Buffering

All digital video players must buffer a certain amount of the video data stream before video playback can occur. These buffering operations appear to the user as a delay between the time the user selects the video to play, and the time when the video begins to play. The **Video** node class includes special internal node instances to indicate to the user that the video data stream is buffering, whether before initial playback, or after buffering must occur again because the video playback is interrupted by an unexpected slowdown in the video stream transfer rate. (Video playback interruptions are more likely to occur if you do not take advantage of some type of adaptive bitrate configuration of the video playback, such as using a segmented adaptive video playback scheme such as HLS, as described in **Adaptive Bitrate Formats**.)

To indicate to the user that the video stream is buffering, the **Video** node class includes two instances of an internal **ProgressBar** node, which, if set to be visible, appear at the bottom center of the video playback screen area automatically when video buffering is occurring.

These progress bars are configured in BrightScript by setting the fields of the internal **ProgressBar** nodes as follows:

- `retrievingBar` for the progress bar that appears when the video stream is being buffered prior to initial playback
- `bufferingBar` for the progress bar that appears when the video stream must be buffered again after an unexpected stream transfer rate slowdown

These internal **ProgressBar** nodes have fields for configuring their appearance, such as setting the color of the portion of the bar that indicates the amount of buffering that has been accomplished. These fields must be set in BrightScript. Other **Video** node fields are available for configuring other aspects of the internal **ProgressBar** nodes, such as setting the color of the text of the progress bar. These fields can be set in either BrightScript or XML markup.

To set the internal node fields in BrightScript, the **Video** node object must be available within the scope of the BrightScript function that will set the fields. For example, if a **Video** node was declared and partially defined in XML markup, you must use the `findNode()` function to declare the node object and access the internal retrieving bar node fields:

```
m.video = m.top.findNode("channel_video_node_id")
...
m.video.retrievingBar.internal_node_field = internal_node_field_value
...
```

> Since you should always set the `control` field of a **Video** node in BrightScript, you should always declare a **Video** node object in BrightScript anyway, either by creating the object in BrightScript, or using the `findNode()` function.

Fast Start Media Playback

Available since firmware version 7.2

Both the **Audio** and **Video** node classes also include a special control option to reduce or eliminate the apparent delay before media playback begins (*fast start*). All digital video requires some time after a video is selected to begin playback, and video (and audio) files streamed over HTTP add network transfer rates to this buffering time. You can configure your application to reduce or eliminate this apparent delay to the user by setting the **Video** (or **Audio**) node `control` field to `prebuffer`, at a time when the user has moved focus to a description of the media item, but before the user actually starts the playback. The media stream will buffer in the background while the user is reading the description of the media item. Then, if and when the user actually selects the media item to play, you can set the `control` field to `play` as usual.

For example, if you have designed your SceneGraph application to have a screen that shows a description of a video, with a button to actually begin playback, you can use the `prebuffer` option of the `control` field in a callback function triggered by the screen focus event as follows:

```
sub setDetailsScreenFocus()
  if m.top.isInFocusChain() and not m.buttons.hasFocus() and not
m.videoPlayer.hasFocus() then
    m.buttons.setFocus(true)
    ' prebuffer video while user is reading the details screen
    m.videoPlayer.control = "prebuffer"
  end if
end sub
```

Then write the callback function for the playback button press event that includes:

```
m.videoPlayer.control = "start"
```

If the user has taken a few seconds to read the details screen, the video will start immediately after the playback button is pressed. An sample channel demonstrating Fast Video Start can be found here: FastVideoStart.zip.

MPEG-4 Playback

For MPEG-4 (mp4) video files without segmented adaptive bitrate switching files, set the **ContentNode** node `streamformat` meta-data attribute to `mp4`, and the `url` attribute to the URL of the MPEG-4 video file.

Segmented Video Playback

For the most basic segmented video playback, you only need to set the `URL` and `StreamFormat` field values in a **ContentNode** node, assign the **ContentNode** node to the `content` field of the **Video** node, then set the **Video** node `control` field value to `play` to start the video. For example:

```
videoContent = createObject("RoSGNode", "ContentNode")
videoContent.url = "video_URI"
videoContent.streamformat = "hls"
m.video = m.top.findNode("video_node_ID")
m.video.content = videoContent
m.video.control = "play"
```

Segmented Video Playback with PlayReady DRM

If you include PlayReady DRM in your DASH video stream, set the following **Content Meta-Data** attributes in your **ContentNode** node:

- `encodingtype`
- `encodingkey`
- `streamformat`
- `url`

Configuring Trick Play

For more advanced video playback (for example, incorporating "trick play" DVD-like scene indexing functionality), there are fields in the **Video** node and the associated **ContentNode** node to provide this functionality.

BIF Files Trick Play

To support trick play for a video, you must generate and provide a set of BIF files for the video. See **Trick Mode Support** for information on BIF file generation and usage.

After you have created the BIF files for a video, you must place the files on a server, and then configure the **Video** node to access the files by setting the associated **ContentNode** node using the trick play attributes described in **Content Meta-Data** (see **Content Meta-Data**).

Configuring Trick Play Appearance

The **Video** node class includes an internal **TrickPlayBar** node to provide the user interface for trick play. You can customize the appearance of this internal node by accessing its fields in BrightScript in the same manner as the internal **ProgressBar** nodes (see **Configuring the ProgressBar Nodes**).

Configuring Closed Captions

Adding closed caption support is a simple matter of configuring the subtitleconfig video node content element. See SceneGraphCaptionsDemo.zip f or an example of using side loaded TTML captions in your channel.

Reading and Assigning Video Configuration Content Meta-Data from your Server

Generally you will want to have all of your content meta-data configuration of a particular video in an XML/JSON file on your server. You can then read this file, convert it to a **ContentNode** node, and set up the related user interface and video configuration for your video in your Roku application.

This XML/JSON file should contain the strings or links to all the information you need to present the video to the user for selection, and to configure the video playback. In almost all cases, you should be able to use the attributes listed in **Content Meta-Data** (see **Content Meta-Data**), which are all recognized as field names for a **ContentNode** node. In the rare cases where you require a custom attribute not found in **Content Meta-Data**, you can create a custom **ContentNode** node by adding **<interface>** fields, or possibly by setting up a parallel associative array that can be accessed using the same user interface element node indexing that you use for accessing **ContentNode** nodes.

Descriptive Content Meta-Data

Descriptive content meta-data attributes are used to communicate information to allow the user to select a video for playback. Most of the attributes are either optional, or are specific to a particular type of video. Set these attributes as you want for your user interface design, and the types of videos you offer. The descriptive meta-data attributes are:

- `contenttype`
- `title`
- `titleseason`
- `description`
- `watched`
- `length`
- `releasedate`
- `rating`
- `starrating`
- `userstarrating`
- `shortdescriptionline1`
- `shortdescriptionline2`
- `episodenumber`
- `numepisodes`
- `actors`
- `actor`
- `directors`
- `director`
- `categories`
- `category`
- `hdbranded`
- `ishd`
- MPAA and TV Ratings (icon identifiers)

If you participate in the **Roku Search** program, you can use the information in the XML feed you supply to Roku for many of the meta-data attributes listed above. See **Roku Search** for complete details.

Video Configuration Meta-Data

See **Content Meta-Data** for the complete list of video configuration meta-data.

Example Video Meta-Data XML Task Node File Reader

The following is an example of a **Task** node that reads an XML file from a server containing descriptive and configuration meta-data for videos. The **Task** node converts the XML file attributes for each video, and builds the corresponding **ContentNode** node with the attribute data. The **Task** node should be created, have an observer callback set for the `videocontent` **<interface>** field, configured with the URL of the server XML file as the `metadatauri` **<interface>** field, then run. When the **ContentNode** node is complete, it is assigned to the `videocontent` **<interface>** field of the **Task** node, which then triggers the callback function to configure lists or grids for each video item, and allow the configuration meta-data to be assigned to a **Video** node that plays the video.

Example Video Content Meta-Data Task Node

```
<component name = "MetaDataCR" extends = "Task" >

  <interface>
    <field id = "metadatauri" type = "uri" value = "" />
    <field id = "videocontent" type = "node" />
  </interface>

  <script type = "text/brightscript" >

    <![CDATA[

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

    sub getContent()
      videocontent = createObject("RoSGNode","ContentNode")
      metadataxml = createObject("roXMLElement")

      ' uncomment/conditionalize for development package XML transfers
(pkg:/server/foo.xml)
      xmlstring = ReadAsciiFile(m.top.metadatauri)
      metadataxml.parse(xmlstring)

      ' uncomment/conditionalize for published channel Internet XML transfers
(http://serverdomain/foo.xml)
      ' readInternet = createObject("roUrlTransfer")
      ' readInternet.setUrl(m.top.metadatauri)
      ' metadataxml.parse(readInternet.GetToString())

      if metadataxml.getName()="MetaData"
        for each video in metadataxml.GetNamedElements("video")
          videoitem = videocontent.createChild("ContentNode")
          videoItem.setFields(video.getAttributes())
        end for
      end if
      m.top.videocontent = videocontent
    end sub

    ]]>

  </script>

</component>
```

Sample Channels

The table below summarizes all of the downloadable samples demonstrating the video playback features introduced in this section.

Sample Channel	Description
Simple_Grid_and_Video.zip	Video player in a simple grid UI.
Simple_Grid_with_Details_and_Video.zip	Video player in a simple grid UI with details page.
PanelSet_and_Video.zip	Video player in a simple PanelSet UI.
VideoList.zip	Video player with content selected from a LabelList.
FastVideoStart.zip	Sample demonstrating how to use Fast Video Start.
SceneGraphCaptionsDemo.zip	Sample demonstrating how to integrate closed caption support.