Roku Object Reference

Matches HD2000 Software Version: 2.0.94

BrightSign™

California, USA
www.roku.com/brightsign

Table of Contents

Introduction

Roku Objects (RO) are the standardized way Roku software exposes functionality for our products' public SDKs. In other words, to publish a new API, Roku will create a new Roku Object. The first product to use this method is the BrightSign.

Roku Objects have these key objectives:

- To be largely language independent.
- To be robust to software upgrades. A RO interface, once established, never changes its methods or its binary linkage.
- To be compatible with compiled or interpreted languages. ROs are completely discoverable and callable at run time or compile time.
- To support multiple abstract interfaces. This allows objects to be used in powerful ways as we'll see below.

As well as the core Roku Object architecture, this reference also defines event architecture and the required interfaces to participate in this scheme. The event scheme is a fundamental piece of any multi-threaded application. Its standardization is the first part in a sequence of RO based standards Roku will adopt across our products, to maximize compatibility of third party plug-ins.

This document describes the Roku Object architecture as two main sections:

- how to use them (as a script writer)
- the initial objects defined for BrightSign

Roku Object Interfaces and Methods

Every Roku Object consists of one or more "Interfaces". An RO Interface consists of one or more Methods. For example, the roVideoPlayer has two interfaces: ifMediaTransport and ifSetMessagePort. The Interface ifSetMessagePort has one Member: SetPort.

For example:
```
p = CreateObject("roMessagePort")
video= CreateObject("roVideoPlayer")

gpio = CreateObject("roGpioControlPort")
gpio.SetPort(p)
video.SetPort(p)
```

This syntax makes use of a short cut provided by the language: The interface name is optional, unless it is needed to resolve name conflicts.

For example:
```
gpio.SetPort(p)
```

is the same as:
```
gpio.ifSetMessagePort.SetPort(p)
```

Note that the abstract Interface ifSetMessagePort is exposed and implemented by both the roGpioControlPort and the roVideoPlayer objects. Once the method SetPort is called, these objects will send their events to the supplied message port. This is discussed more in the Event section below.

Once an interface is defined and published, it is never changed. For example, imagine if Roku decided in a future release that the ifSetMessagePort really needed another method, say "ClearMessagePort". Instead of changing the ifSetMessagePort interface, we would create a new interface ifSetMessagePort2. This interface would contain only the new methods. The "old" method still exists in the original Interface. Older applications will only use the original interface, but newer applications could use the original and the new interface. This ensures full backwards compatibility with the installed base of Roku scripts and applications with future versions of Roku products.

Roku Objects consist only of interfaces. Interfaces define only methods. There is no concept of a "property" or variable at the Object or Interface level. These must be implemented as Set/Get methods in an Interface.

Inheritance

There is no explicit support for Inheritance in the RO architecture.   However, this section is a brief discussion of how C++ concepts of inheritance map to ROs.

- **Use of Virtual Base classes to abstract interfaces**.   For example, in C++ one might create a virtual base class for a *AsciiStream* Object. Then create implementation class for an *RS232Port*, *TCP*, or *Keyboard*.   This type of functionality is accomplished with Roku Objects by defining an Interface (for example *ifAsciiStream*), then ROs that expose this interface (e.g. the *roTCP*)
- **Use of "Mix-in" classes to bring-in existing functionality**.   ROs don't have an exact equivalent. If the writer of an object wants to bring-in existing RO they can create them and hold a reference to them internally. If the object creator wants to expose a created objects Interface(s), it must expose the same interface (all interfaces are public), and then when an interface method is called, the RO calls the appropriate Interface of a called object.

Classes

A Class Name is the name used to create a Roku Object. For example:

```
video= CreateObject("roVideoPlayer")
```

roVideoPlayer is the class name.

Object and Class Name Syntax

Class names:

- must start with an alphabetic character (a – z)
- may consist of alphabetic characters, numbers, or the symbol   "_" (underscore)
- they are not case sensitive
- may be of any reasonable length

Types

The following types are currently defined for values that can be passed to or returned from a RO Method:

- "rotVOID"
- "rotINT32"
- "rotFLOAT"
- "rotSTRING"
- "rotBOOL"
- "rotOBJECT"
- "rotINTERFACE"

Note that:

- Types are strings.
- The class name of an object is the type of that object.

Zones (HD2000 only)

With the BrightSign Zones feature, you can divide the screen into rectangles and play different content in each rectangle.

A zone can contain video and images, images, a clock, or text. There can be only one video zone per screen. However, there can be multiple of other types of zones on the screen. A text zone can contain simple text strings or can be configured to display an RSS feed in a ticker type display.

To enable zone functionality, the following global function must be called in the script:

```
rotVOID EnableZoneSupport(rotBOOL enable)
```

When zones are enabled, the image layer is always on top of the video layer. When zones are not enabled, the image layer is hidden whenever video is played, and the video layer is hidden whenever images are played.

Example:
This script creates 5 zones. The first one contains a video player, the next two contain image players, the fourth has a clock widget and the last is a text widget. The objects used in this example are explained in subsequent sections of this document.

```
debug = true

v=CreateObject("roVideoMode")
v.SetMode("1920x1080x60i")
EnableZoneSupport(true)

r=CreateObject("roRectangle", 60, 60, 1386, 800)
v=CreateObject("roVideoPlayer")
v.SetRectangle(r)
v.SetLoopMode(1)
v.SetViewMode(2)
v.SetAudioOutput(2)
v.PlayFile("Amazon_1080.mpg")

i1=CreateObject("roImagePlayer")
i1.SetDefaultMode(3)
r=CreateObject("roRectangle", 100, 100, 285, 123)
i1.SetRectangle(r)
i1.DisplayFile("splash_master.png")

i2=CreateObject("roImagePlayer")
i2.SetDefaultMode(2)
r=CreateObject("roRectangle", 1520, 155, 300, 300)
i2.SetRectangle(r)
ok=i2.DisplayFile("museum_ad1.jpg")
if ok=0 then
    i2.DisplayFile("museum_ad1.jpg")
endif

i3=CreateObject("roImagePlayer")
i3.SetDefaultMode(2)
r=CreateObject("roRectangle", 1520, 565, 300, 300)
i3.SetRectangle(r)
ok=i3.DisplayFile("museum_ad2.jpg")
if ok=0 then
```

```
    i3.DisplayFile("museum_ad2.jpg")
endif

r=CreateObject("roRectangle", 1520, 50, 300, 100)
res=CreateObject("roResourceManager", "resources.txt")
c=CreateObject("roClockWidget",r, res, true)
c.Show()

r=CreateObject("roRectangle", 60, 900, 1800, 100)
t=CreateObject("roTextWidget", r, 3, 0, 5)
t.SetForegroundColor(&ha0a0a0)
t.PushString("The next Museum tour will be starting at 2:30pm in
the lobby.")
t.PushString("Visit the Museum Store today for 15% all
purchases.")
t.PushString("Become a museum member and today's visit is free.")
t.PushString("BrightSign solid-state media players power
interactive video exhibits with simplicity, reliability and
interactivity to high-impact digital signage. Learn more at
www.roku.com/brightsign.")
t.Show()

p=CreateObject("roMessagePort")

sw = CreateObject("roGpioControlPort")
sw.SetPort(p)

msg_loop:
    msg=wait(0,p)

    if type(msg)="roGpioButton" then
        if debug AND msg=12 then 'Video Select
            end
        endif
      else
            print "Unknown event "; type(event)
      endif

    goto msg_loop
```

BrightSign Object Library

This section specifies each of the Roku Objects that are included with BrigthScript.

Event Loops

When creating anything more than a very simple script, an Event Loop will need to be created.   An Event Loop typically has this structure:

1. wait for the event
2. process the event
3. jump back to 1

Events are things like a button press, a timer that has triggered, a video that has finished playing back, etc.

By convention, Roku Object (RO) events work as follows.

- A RO of type “roMessagePort” is created. In BrightScript, by the user’s script.
- ROs that can send events are instructed to send their events to this message port. You could set up multiple message ports, and have each event go to its own message port, but it is usually simpler to just create one message port, and have the events all go to this one port. To instruct the RO to send events to a specific port, use the *ifSetMessagePort* Interface.
- The script waits for an event. The actual function to do this is the ifMessagePort.WaitMessage(), but if you are using BrightScript, the built-in statement WAIT makes this easy.
- If multiple event types are possible, your script should determine which event that the wait received, then process it. The script then jumps back to the Wait.

An “Event” can be generated by any Roku Object. For example, the class “roGpioControlPort” sends events of type “roGpioButton”. The “roGpioButton” has one interface: ifInt. ifInt allows access to an integer. An event loop needs to be aware of the possible events it can get, and process them.

Example

```
print "BrightSign Button-LED Test Running"
p =   CreateObject("roMessagePort")
tmr = CreateObject("roMessagePort")
gpio =  CreateObject("roGpioControlPort")
gpio.SetPort(p)

event_loop:
   msg=wait(0, p)
   if type(msg)<>"roGpioButton" then event_loop
   butn = msg.GetInt()
   if butn > 5 then event_loop
   gpio.SetOutputState(butn+17,1)
   print "Button Pressed: ";butn
   msg = wait (500, tmr)
   gpio.SetOutputState(butn+17,0)

   clear_events:
       msg=p.GetMessage():if type(msg)<>"rotINT32" then clear_events
   goto event_loop
```

Classes

For each class a brief description is given, a list of interfaces, and the member functions in the interfaces.

roList

A general purpose doubly link list. It can be used as a container for arbitrary length lists of Roku Objects.

Interfaces:

- *ifList*

```
rotINT32 Count(rotVOID)
rotBOOL IsEmpty(rotVOID)
rotBOOL ResetIndex(rotVOID)
rotVOID AddTail(rotOBJECT obj)
rotVOID AddHead(rotOBJECT obj)
rotBOOL FindIndex(rotSTRING name)
rotOBJECT RemoveIndex(rotVOID)
rotOBJECT GetIndex(rotVOID)
rotOBJECT RemoveTail(rotVOID)
rotOBJECT RemoveHead(rotVOID)
rotOBJECT GetTail(rotVOID)
rotOBJECT GetHead(rotVOID)
```

roMessagePort

A message port is the place messages (events) are sent. See the “Event Loops” section for more details. When using Roku BrightScript, you would not call these functions directly. Instead, use the “Wait” BrightScript statement (see the BrightScript documentation).

Interfaces:

- *ifMessagePort*

```
rotOBJECT GetMessage (rotVOID)
rotOBJECT WaitMessage(rotINT timeout)
rotVOID   PostMessage(rotOBJECT msg)
```

roVideoMode

This class allows you to set the output video resolution. The same video resolution is applied to all video outputs on BrightSign. Video or images that are subsequently decoded and displayed will be scaled (using the hardware scalar) to this output resolution if necessary.

Interfaces:

- *ifVideoMode*
  `rotBOOL SetMode(rotSTRING mode)`

- *ifVideoMode*
  `rotINT32 GetResX()`

- *ifVideoMode*
  `rotINT32 GetResY()`

- *ifVideoMode*
  `rotINT32 GetSafeX()`

- *ifVideoMode*
  `rotINT32 GetSafeY()`

- *ifVideoMode*
  `rotINT32 GetSafeWidth()`

- *ifVideoMode*
  `rotINT32 GetSafeHeight()`

- *ifVideoMode*
  `rotBOOL SetPowerSaveMode(rotBOOL power_save_enable)`

- *ifSetMessagePort*
  `rotVOID SetPort(rotOBJECT obj)`

Supported modes that can be passed to SetMode on the HD600 are:

- "ntsc"
- "pal"
- "640x480x60p"
- "800x600x60p"
- "1024x768x60p"
- “1280x768x60p”
- "720x480x60p"
- "1280x720x60p"

Supported modes that can be passed to SetMode on the HD2000 are:

- "1024x768x60p"
- "720x480x60p"
- “720x576x50p”
- "1280x720x60p"
- “1280x720x50p”
- “1280x768x60p”
- “1920x1080x60i”

GetResX, GetResY (HD2000 only currently)

Get the total display size for the current video mode.

GetSafeX, GetSafeY (HD2000 only currently)

Get the left and top coordinates for the start of the "safe area". For modes that are generally displayed with no overscan, both will be zero.

GetSafeWidth, GetSafeHeight (HD2000 only currently)

Get the width and height of the "safe area". For modes that are generally displayed with no overscan, these will return the same as GetResX and GetResY.

More information about safe areas can be found at:
http://en.wikipedia.org/wiki/Safe_area and
http://en.wikipedia.org/wiki/Overscan_amounts

SetPowerSaveMode (HD2000 only currently)

Turns off the syncs for VGA output and the DAC output for component video. For some monitors, this will cause the monitor to go into standby mode.

Note that the BrightSign Hardware has a video anti-aliasing low pass filter that is set automatically. See the hardware manual for more information.

On the HD2000, if the video mode specified in SetMode is different than the object's current video mode, the unit will reboot and set the unit's video mode to the new setting during system initialization.

Example:
This script prints out video mode resolutions as the user presses the front panel button (note that the video selector front panel button is not currently implemented on the HD2000):

```
v=CreateObject("roVideoMode")
v.SetMode(“1024x768x60p”)
p=CreateObject("roMessagePort")
v.SetPort(p)

loop:
msg=wait(0,p)
print msg.GetWidth()
print msg.GetHeight()
goto loop
```

roVideoModeChange

If SetPort is called, messages that are of type roVideoModeChange will be sent to the port and has the following member functions:

- `rotINT32 GetWidth()`
- `rotINT32 GetHeight()`

roVideoPlayer

A Video Player is used to play back video files (using the generic ifMediaTransport Interface). If the message port is set, the object will send events of type roVideoEvent. All object calls are asynchronous. That is, video playback is handled in another thread from the script. The script will continue to run while video is playing. Note that the HD600 will decode a maximum resolution of D1 (DVD Quality). HiDef video is not able to be decoded. However, the HD2000 will decode to a maximum resolution of 1920x1080 interlaced, including HiDef video. The decoded video will be scaled to the output resolution specified by roVideoMode.

NOTE:

- Currently only MPEG2 files are supported by BrightSign
- All HD600 video files must have a 48Khz audio track, even if it is silence. This restriction does not apply to the HD2000.
- Currently only MPEG2 program streams are supported. Simple elementary video streams are not supported. A video stream must be muxed into a program stream. Audio can be either AC3 or PCM.

Interfaces:

- *ifSetMessagePort*
  ```
  rotVOID SetPort(rotOBJECT obj)
  ```

- *ifAudioControl – see roAudioPlayer for docs*

- *ifVideoControl*
  ```
  rotBOOL SetViewMode(rotINT32 mode)
  rotVOID SetRectangle(roRectangle r)
  ```

- *ifMediaTransport*
  ```
  rotBOOL PlayFile(rotSTRING filename)
  rotBOOL Stop(rotVOID)
  rotBOOL Play(rotVOID)
  rotBOOL SetLoopMode(rotINT32 mode)
  rotBOOL ClearEvents(rotVOID)
  rotBOOL AddEvent(rotINT32 userdata, rotINT32 time_in_ms)
  rotBOOL StopClear()
  ```

  If you wish to use a view mode different from the default, it must be set prior to starting video playback.

  view_mode values:
  0 - Scale to fill (default). The aspect ratio can alter.
  1 - Letterboxed and centered. The aspect ratio is maintained and the video has black borders.
  2 - Fill screen and centered. The aspect ratio is maintained and the screen is filled.

To display the video in a zone, `SetRectangle`() must be called. `EnableZoneSupport`() must be called to use zones functionality.

MPEG2 video files are encoded with a specific aspect ratio, and output display resolutions have an aspect ratio. Video display modes 1 and 2 use these aspect ratios to ensure that the video file aspect ratio is preserved when it is displayed. The only time that this will fail is when a widescreen monitor displays a 4:3 output resolution such as 800x600 across the whole screen i.e. the monitor doesn't respect the aspect ratio. Please note that this feature relies on the correct aspect ratio marking of the MPEG2 video files. Unfortunately, not all files are marked correctly.

Users can add events which trigger messages of the roVideoEvent “Timecode Hit” at the specified millisecond times in a video file. The data field of the roVideoEvent holds the userdata passed in with AddEvent.

Here is an example script that uses timecode events. The script prints out 2, 5 and 10 at 2 seconds, 5 seconds and 10 seconds into the video. The msg is approaching frame accurate.

```
10 v = CreateObject("roVideoPlayer")
20 p = CreateObject("roMessagePort")
30 v.SetPort(p)
40 ok = v.AddEvent(2, 2000)   ' Add timed events to video
50 ok = v.AddEvent(5, 5000)
60 ok = v.AddEvent(10, 10000)
70 ok = v.AddEvent(100, 100000)
80 ok = v.PlayFile("ATA:/C5_d5_phil.vob")
90 msg = wait(0,p)                     ' Wait for all events
95 if msg.GetInt() = 8 then stop        ' End of file
100 if msg.GetInt() <> 12 goto 90       ' I only care about time events
110 print msg.GetData()    ' Print out index when the time event happens
120 goto 90
```

roAudioPlayer

An audio player is used to play back audio files (using the generic ifMediaTransport Interface). If the message port is set, the object will send events of type roAudioEvent. All object calls are asynchronous. That is, audio playback is handled in another thread from the script. The script may continue to run while audio is playing.

NOTE:

- Typically filenames must start with a "/" (e.g. "/mymusic.mp3").
- MP3 files are supported on all BrightSigns while the HD2000 also supports WAV files.

Interfaces:

- *ifSetMessagePort*
  `rotVOID SetPort(rotOBJECT)`

- *ifMediaTransport*
  `See roVideoPlayer for docs`

- *ifAudioControl*

```
      rotBOOL SetAudioOutput(rotINT32 audio_output)
      rotBOOL SetAudioMode(rotINT32 audio_mode)
      rotBOOL MapStereoOutput(rotINT32 mapping)
      rotBOOL SetVolume(rotINT32 volume)
      rotBOOL SetChannelVolumes(roINT32 channel_mask, roINT32 volume)
      rotBOOL SetAudioOutputAux(rotINT32 audio_output)
      rotBOOL SetAudioModeAux(rotINT32 audio_mode)
      rotBOOL MapStereoOutputAux(rotINT32 mapping)
      rotBOOL SetVolumeAux(rotINT32 volume)
      rotBOOL SetChannelVolumesAux(roINT32 channel_mask, roINT32
volume)
      rotBOOL SetAudioStream(rotINT32 stream_index)
      rotBOOL SetAudioStreamAux(rotINT32 stream_index)
```

Before changing the audio output when a video file is playing or has played, a call to video.Stop() is needed.

audio_output values:
0 - Analog audio
1 - USB audio
2 - SPDIF audio, stereo PCM
3 - SPDIF audio, raw AC3
4 - analog audio with SPDIF mirroring raw AC3

audio_mode values
(Options 0 and 1 only apply to video files; 2 applies to all audio sources)
0 - AC3 Surround
1 - AC3 mixed down to stereo
2 - No audio

mapping values
(used to select which analog output if audio_output set to 0)
0 - Stereo audio is mapped AUDIO-3

1 - Stereo audio is mapped to AUDIO-2
2 - Stereo audio is mapped to AUDIO-1

set_volume
Volume is a percentage and so takes a value 0-100. The volume value is clipped prior to use i.e. SetVoume(101) will set the volume to 100 and return TRUE. The volume is the same for all mapped outputs and USB/SPDIF/analog. There is however a separate volume level stored for audioplayer and videoplayer.

Set_channel_volumes
You can control volume on individual audio channels. This volume command takes a hex channel mask which determines which channels to apply the volume to and a level which is a percentage of full scale. The volume control works on the channel of audio rather than the output. The channel mask is a bit mask with the following bits for AC3 output:

&H01	Left
&H02	Right
&H04	Center
&H08	Subwoofer
&H10	Left surround
&H20	Right surround

&H3f is all channels, &H07 is just the LCR channels (Left, Center, Right), &H03 would just be right and left, &H30 just the surrounds, etc. The channels are the channels in the audio file and not the output channels i.e. if you are playing a stereo file but have mapped it to the middle analog output then its volume is still controlled by the Left and Right bits &H01 and &H02.

Example: This code sets audio output to come out the Audio 1 port:
```
video = CreateObject("roVideoPlayer")
video.SetAudioMode(1)      ‘ STEREO
video.SetAudioOutput(0)
video.MapStereoOutput(2)
```

Example: This code sets audio output to come out USB port to a USB Speaker
```
video.SetAudioMode(0)    ‘ SURROUND 5.1 decoder
video.SetAudioOutput(1)  ‘ USB
```

Example: This code sets the volume level for individual channels
```
audio = CreateObject(“roAudioPlayer”)
audio.SetChannelVolumes(&H01, 60)   ‘left channel to 60%
audio.SetChannelVolumes(&H02, 75)   ‘right channel to 75%
audio.SetChannelVolumes(&H04, 80)   ‘center channel to 80%
audio.SetChannelVolumes(&H07, 70)   ‘left, right, center channel to 70%
audio.SetChannelVolumes(&H3f, 65)   ‘all channels to 65%
```

Playing Multiple Audio Files Simultaneously (HD2000 only)
Multiple MP3 files along with an audio track as part of a video file can be played to any combination of the following:
- Analog outputs 1, 2, or 3
- SPDIF / HDMI
- USB

Only a single file can be sent to an output at any given time. For example, two roAudioPlayers cannot simultaneously play to the SPDIF output - the second one to attempt a PlayFile will get an error. To free an output, the audio or video stream must be stopped (using ifMediaTransport's Stop or StopClear calls).

Notes on this functionality:

- SPDIF and HDMI are the same audio output.
- Currently only a single set of USB speakers is supported.
- Only similar files may be played out of analog outputs simultaneously (meaning that it is not possible to play a WAV file out of one analog output while simultaneously playing an MP3 file out of another analog output).
- If the audio for a playing video is connected to one of the analog outputs, any remaining analog outputs cannot be used by other sources. That is, the remaining analog outputs cannot be used for MP3 or WAV file playback.
- Each audio and video stream played consumes some of the finite CPU resources. The amount consumed depends on the bitrates of the streams. Testing is the only way to really tell whether a given set of audio files can be played at the same time as a video

Example: This code plays a video with audio over SPDIF/HDMI and an mp3 file to an analog port:

```
video=CreateObject("roVideoPlayer")
video.SetAudioMode(0)
video.SetAudioOutput(3)
video.PlayFile("video.mpg")

audio=CreateObject("roAudioPlayer")
audio.SetAudioOutput(0)
audio.MapStereoOutput(2)
audio.PlayFile("audio.mp3")
```

Playing Video Files with Multiple Audio Streams (HD2000 only)

MPEG-2 program streams can optionally contain a number of audio streams. These can all be the same or different formats. Generally this is used either to provide alternative audio formats, or alternative mixes/languages.

The HD2000 supports the playback of two audio streams from within a single program stream. The two audio streams may be played back simultaneously (subject to the limitations documented above for playing back multiple audio files simultaneously), or a specific audio stream from within a program stream can be selected to support multiple languages / mixes.

The following functions control the auxiliary stream:

- SetAudioOutputAux (audio_output)
- SetAudioModeAux (audio_mode)
- MapStereoOutputAux (mapping)
- SetVolumeAux (volume)
- SetChannelVolumesAux (channel_mask, volume)

The following functions control which specific streams are played

- SetAudioStream (stream_index)
- SetAudioStreamAux (stream_index)

The stream_index is the index of the audio stream within the program stream to be played. A value of -1 indicates that the first audio stream found within the multiplex will be played.

Example: This code plays a video with the main audio stream over SPDIF/HDMI and the auxiliary stream to an analog port:

```
video=CreateObject("roVideoPlayer")
```

```
video.SetAudioStream(0)
video.SetAudioOutput(3)
video.SetAudioStreamAux(1)
video.SetAudioOutputAux(0)
video.PlayFile("TwoStreams.mpg")
```

Example: This code shows how to play a video with multiple languages:

```
REM – select language from first audio track
video.SetAudioStream(0)
video.PlayFile("Language.mpg")

REM – select language from second audio track
video.SetAudioStream(1)
video.PlayFile("Language.mpg")
```

Because the streams to be played are specified prior to playing the file, there are some rules necessary as to what happens if the streams selected are invalid:

- If the main stream is set to -1 then the first audio stream found within the multiplex will be played.
- If the main stream is not -1 but is invalid, then no stream will be played. The aux stream will also be disabled in this case.
- If the aux stream is -1 or invalid, then no audio will be played out of the aux output.

roVideoEvent() and roAudioEvent()

Video and Audio events can have one of these integer values. They are declared as separate classes as they are likely to diverge in the future.

0 Undefined	Player is in an undefined state.
1 Stopped	Playback of the current media item is stopped.
3 Playing	The current media item is playing.
4 ScanForward	The current media item is fast forwarding.
5 ScanReverse	The current media item is fast rewinding.
6 Buffering	The current media item is getting additional data from the server.
7 Waiting	Connection is established, but the server is not sending data. Waiting for session to begin.
8 MediaEnded	Media item has completed playback.
9 Transitioning	Preparing new media item.
10 Ready	Ready to begin playing.
11 Reconnecting	Reconnecting to stream.
12 TimeHit	A particular timecode is hit. See roVideoPlayer.

Interfaces:

- *ifInt – contains event id enumerated above*
  `rotINT32 GetInt(rotVOID)`

- *ifData – contains userdata*
  `rotINT32 GetData(rotVOID)`

Example Code Clip:
```
    vp_msg_loop:
        msg=wait(tiut, p)
        if type(msg)="roVideoEvent" then
            if debug then print "Video Event";msg.GetInt()
            if msg.GetInt() = 8 then
                if debug then print "VideoFinished"
                retcode=5
                return
            endif
        else if type(msg)="roGpioButton" then
            if debug then print "Button Press";msg
            if escm and msg=BM then retcode=1:return
            if esc1 and msg=B1 then retcode=2:return
            if esc2 and msg=B2 then retcode=3:return
            if esc3 and msg=B3 then retcode=4:return
        else if type(msg)="rotINT32" then
            if debug then print "TimeOut"
            retcode=6
            return
        endif

        goto vp_msg_loop
```

roGpioControlPort

This object is used to control and wait for events on the BrightSign generic DB25 control port and front panel. Typically LEDs or Buttons are connected to the DB25 port.
Turning on a GPIO output puts the voltage on the GPIO port to 3.3V. Turning off a GPIO output puts the voltage on the GPIO port to 0 V.

The output ids are as follows:

> Front panel LEDs start at id 0 on the left and go up to id 16 on the right
> DB25 GPIO outputs start at id 17 and go up to id 22.
> Note: SetWholeState will overwrite any prior output settings.
> SetOutputState takes an output id (1, 2, or 17 for example.)
> SetWholeState takes a mask – for example SetWholeState(2^1 + 2^2 + 2^17) to set ids 1,2, and 17.

The input ids are as follows

> DB25 GPIO inputs start at id 0 and go up to id 11
> Front panel switch is id 12

Interfaces:

- *ifSetMessagePort*
  ```
  rotVOID SetPort(rotOBJECT obj)
  ```

- *ifGpioControlPort*
  ```
  rotBOOL IsInputActive(rotINT32 input_id)
  rotINT32 GetWholeState(rotVOID)
  rotVOID SetOutputState(rotINT32 output_id, rotBOOL onState)
  rotVOID SetWholeState(rotINT32 on_state)
  ```

roGpioButton

Interfaces:

- *ifInt – contains input id listed above*

  ```
  rotINT32 GetInt(rotVOID)
  ```

roQuadravoxSNS5

This object is used to control and wait for events on the Quadravox SNS5 serial button/LED box.
Interfaces:

- *ifSetMessagePort*
  ```
  rotVOID SetPort(rotOBJECT)
  ```

- *ifQuadravoxSNS5 – similar to ifGpioControlPort but with the addition of a Flash state*
  ```
  rotBOOL IsInputActive(rotINT32 id)
  rotINT32 GetWholeState(rotVOID)

  rotVOID SetOutputState(rotINT32 id, BOOL on_state,
                          BOOL flash_state)
  rotVOID SetWholeState(rotINT32 on_state, rotINT32 flash_state)
  rotVOID SetFlashRate(rotINT32 flash_rate)
  ```

Notes on flash rate: The range is from 0 to 31, 0 is fast, 31 is slow. The default is 2.

roQuadravoxButton

Similar to roGpioButton except that it originates from the Quadravox
Interfaces:

- *ifInt – contains button id*

  ```
  rotINT32 GetInt(rotVOID)
  rotVOID SetInt(rotINT32 id)
  ```

roKeyboard

This object is used to wait for events from a USB keyboard.

Interfaces:

- *ifSetMessagePort*
  `rotVOID SetPort(rotOBJECT)`

roKeyboardPress

A keyboard event resulting from the user pressing a key on the USB keyboard. The int value is the ASCII code of the key that was pressed.

Interfaces:

- *ifInt – contains ASCII value of key press*
  `rotINT32 GetInt(rotVOID)`

The rotINT32 returned can have one of the following values:

Letter Keys		**Number Keys**	**Function Keys**	**Misc Keys**	**Special Keys**			
A - 97	R - 114	0 - 48	F1 - 32826	Del - 127	"-"	45	:	58
B - 98	S - 115	1 - 49	F2 - 32827	Backspace - 8	"="	61	"	34
C - 99	T - 116	2 - 50	F3 - 32828	Tab - 9	\	92	<	60
D - 100	U - 117	3 - 51	F4 - 32829	Enter - 13	`	96	>	62
E - 101	V - 118	4 - 52	F5 - 32830	Print Scrn - 32838	[	91	?	63
F - 102	W - 119	5 - 53	F6 - 32831	Scrl Lock - 32839	]	93	!	33
G - 103	X - 120	6 - 54	F7 - 32832	Pause/Brk - 32840	;	59	@	64
H - 104	Y - 121	7 - 55	F8 - 32833	INS - 32841	" ' "	39	#	35
I - 105	Z - 122	8 - 56	F9 - 32834	Home - 32842	,	44	$	36
J - 106		9 - 57	F11 - 32836	Page Up - 32843	.	46	%	37
K - 107			F12 - 32837	Page Down - 32846	/	47	^	94
L - 108				End - 32845	_	95	&	38
M - 109				Caps - 32811	"+"	43	*	42
N - 110				Left Arrow - 32848	|	124	(	40
O - 111				Right Arrow - 32847	~	126	)	41
P - 112				Up Arrow - 32850	{	123		
Q - 113				Down Arrow - 32849	}	125		

roIRRemote

The key code from Roku's custom code using the NEC protocol is decoded and sent as an event.   Use this object to register your Event port.

Interfaces:

- *ifSetMessagePort* interface:
  ```
  rotVOID SetPort(rotObject message_port_object)
  ```

roIRRemotePress

Messages are generated on Roku Soundbridge remote key presses. These have the ifInt interface with the useful function:

Interfaces:

- *ifInt – contains keycode*

  ```
  rotINT32 GetInt(rotVOID)
  ```

The rotINT32 returned can have one of the following values:

West	0
East	1
North	2
South	3
Select	4
Exit	5
Power	6
Menu	7
Search	8
Play	9
Next	10
Previous	11
Pause	12
Add	13
Shuffle	14
Repeat	15
Volume up	16
Volume down	17
Brightness	18

roImagePlayer

Display static bitmap images on the video display.

Interfaces:

- ifImageControl

```
rotBOOL DisplayFile(rotSTRING image_filename)
rotBOOL PreloadFile(rotSTRING filename)
rotBOOL DisplayPreload()
rotBOOL StopDisplay()   // removes an image from the display
rotBOOL DisplayFileEx(rotSTRING filename, rotINT32 mode,
                      rotINT32 x, rotINT32 y)
rotBOOL PreloadFileEx(rotSTRING filename, rotINT32 mode,
                      rotINT32 x, rotINT32 y)

rotBOOL SetDefaultMode(rotINT32 mode);
rotBOOL SetDefaultTransition(roINT32 transition)
rotVOID SetRectangle(roRectangle r)
```

The simplest way to use roImagePlayer is to just make calls to “DisplayFile”. Or you can use PreloadFile()/DisplayPreload() to have more control.

PreloadFile loads the file into memory into an off-screen buffer. DisplayPreload then displays the image in memory to the screen using the on-screen buffer. There are only two memory buffers, one is displayed on screen, the other can be used for preloading. PreloadFile can be called multiple times before DisplayPreload is called and will keep loading into the same off-screen buffer. DisplayFile does a PreloadFile followed immediately by a DisplayPreload, so any previously preloaded image will be lost. If no image is preloaded DisplayPreload will have no effect.

`X&Y:`
x and y indicate which position of the image to center as near as possible, or can both be set to -1, which means to use the center of the image as the point to position nearest the center.

SetDefaultMode sets the mode used for DisplayFile and PreloadFile. If this isn't called the mode is 0 which is centered with no scaling.

image_filename currently must point to a 8-bit, 24-bit, or 32-bit .BMP file. In addition, PNG and JPEG files are supported on the HD2000.

Display Modes supported are:

- 0 - Center image. No scaling takes place, only cropping if the image is bigger than the screen.
- 1 - Scale to fit. The image is scaled so that it is fully viewable with its aspect ratio maintained.
- 2 - Scale to fill and crop. The image is scaled so that it totally fills the screen, though with its aspect ratio maintained.
- 3 - Scale to fill. The image is stretched so that it fills the screen and the whole image is viewable. This means that the aspect ratio will not be maintained if it is different to that of the current screen resolution.

SetDefaultTransition sets the transition to be used when the next image is displayed. Transitions available include:

- 0 - No transition, immediate blit
- 1 to 4 - Image wipes from top, bottom, left and right
- 5 to 8 - Explodes from centre, top left, top right, bottom left, bottom right

- 10 to 11 - Venetian blinds vertical and horizontal
- 12 to 13 - Comb effect vertical and horizontal
- 14 - Fade out to background color then back in
- 15 - Fade between current image and new image
- 16 to 19 - Slides from top, bottom, left and right

To display images in a zone, `SetRectangle`() must be called. `EnableZoneSupport`() must be included in a script to use the zones functionality.

Here are some example shell commands you can try to test the different display modes:

```
Roku> image filename.bmp 0
Roku> image filename.bmp 1
Roku> image filename.bmp 2
Roku> image filename.bmp 3

Roku> image filename.bmp 0 0 0
Roku> image filename.bmp 2 0 0
```

This example script uses preloaded images to improve the UI speed when the user hits a key on the keyboard. As soon as the keyboard is hit, then the display switches to the new image which has already been preloaded. The only delay is if the key is hit whilst the image is pre-loading - as soon as the image is loaded, it will then display.

```
i = CreateObject("roImagePlayer")
p = CreateObject("roMessagePort")
k = CreateObject("roKeyboard")
k.SetPort(p)

i.PreloadFile("one.bmp")

loop:
i.DisplayPreload()
i.PreloadFile("two.bmp")
wait(0,p)
i.DisplayPreload()
i.PreloadFile("one.bmp")
wait(0,p)
goto loop
```

roInt, roFloat, roString

The intrinsic types rotINT32, rotFLOAT, and rotSTRING have an object and interface equivalent. These are useful in the following situations:

- When an object is needed, instead of a typed value. For example, roList maintains a list of objects.
- If any object exposes the ifInt, ifFloat, or ifString interfaces, that object can be used in any expression that expects a typed value. For example, in this way an roTouchEvent can be used as an integer whose value is the userid of the roTouchEvent.

Notes:

- If o is an roInt, then the following statements have the following effects
    1. print o ‘ prints o.GetInt()
    2. i%=o ‘ assigns the integer i% the value of o.GetInt()
    3. k=o ‘presumably k is typeOmatic, so it becomes another reference to the roInt o
    4. o=5 ‘this is NOT the same as o.SetInt(5). Instead it releases o, and ‘changes the type of o to rotINT32 (o is typeOmatic). And assigns it to 5.
- When a function that expects a Roku Object as a parameter is passed an int, float, or string, BrightScript automatically creates the equivalent Roku object.

roInt contains one interface:

- *ifInt*
```
rotINT32 GetInt()
rotVOID SetInt(rotINT32 value)
```

roFloat contains one interface:

- *ifFloat*
```
rotFLOAT GetFloat()
rotVOID SetFloat(rotFLOAT value)
```

roString contains one interface:

- *ifString*
```
rotSTRING GetString()
rotVOID SetString(rotSTRING value)
```

Example:
```
BrightScript> o=CreateObject("roInt")
BrightScript> o.SetInt(555)
BrightScript> print o
 555
BrightScript> print o.GetInt()
 555
BrightScript> print o-55
 500
```

Example:
```
BrightScript> list=CreateObject("roList")
BrightScript> list.AddTail(5)
BrightScript> print type(list.GetTail())
```
Note that an integer value of "5" is converted to type "roInt" automatically, because list.AddTail() expects an Roku Object as its parameter.

Yet Another Example. The function ListDir() returns an object roList of roString’s.

```
BrightScript> l=ListDir("/")
BrightScript> for i=1 to l.Count():print l.RemoveHead():next
test_movie_3.vob
test_movie_4.vob
test_movie_1.vob
test_movie_2.vob
```

roTouchScreen

The touch screen object allows you accept events from touch screen panels or Mice. Currently only the Elo USB touch screens or a USB Mouse/Trackball are supported. However, we are always working on more driver support. Contact brightsignsales@rokulabs.com if you have specific touch panel requests.

roTouchScreen responds to clicks with a USB mouse the same way it does to touches on a touch screen. However, you will need to provide a cursor bitmap if you want to enable mouse support. There is one you can use in the Roku BrightSign demo which can be downloaded from our web site.

To use a touch screen follow these general steps:

1. create an roTouchScreen
2. Use SetPort to tell the roTouchScreen which roMessagePort to send events to
3. Define one or more touch regions. A touch region may be rectangular or circular. When someone touches the screen anywhere inside the area of a touch region, an event will be sent to your message port.
4. If touch areas overlap such that a touch hits multiple regions, an event for each region touched will be sent.
5. Process the events.

roTouchScreen supports rollover regions. Rollovers are based around touch regions. When a rectangular or circular region is added it defaults to having no rollover. You can enable a rollover using the touch region's ID and specifying an on and off image. Whenever the mouse cursor is within that region the on image is displayed and the rest of the time the off image is displayed. This allows buttons to highlight as the mouse cursor moves over them.

roTouchScreen has these interfaces

1. `ifTouchScreen`
2. `ifSetMessagePort`

ifTouchScreen has these member functions:

```
rotVOID SetResolution(rotINT32 x, rotINT32 y)
rotVOID AddRectangle_region(rotINT32 x, rotINT32 y, rotINT32 w,
                                rotINT32 h, rotINT32 userid)
rotVOID AddCircleRegion(rotINT32 x, rotINT32 y, rotINT32 radius,
                                rotINT32 userid)
rotVOID ClearRegion()
rotSTRING GetDeviceName()
rotVOID SetCursorPosition(rotINT32 x, rotINT32 y)
rotVOID SetCursorBitmap(rotSTRING, rotINT32 x, rotINT32 y)
rotVOID EnableCursor(rotBOOL on-off)
rotVOID EnableRollover(rotInt32 region_id, rotString on_image,
                       rotString off_image, rotBool cache_image)
rotVOID EnableRegion(rotInt32 region_id, rotBool enabled)
rotVOID SetRollOverOrigin(rotInt32 region_id, rotInt32 x, rotInt32
y)
rotBOOL IsMousePresent(rotVOID)
```

roTouchScreen sends events of type roTouchEvent. roTouchEvent has these interfaces:

1. ifInt (the userid of the touched region)
2. ifPoint (the x,y coordinates of the touch point. Not normally needed). ifPoint has two member functions:
   a. rotINT32 GetX()
   b. rotINT32 GetY()

3. ifEvent (mouse events). ifEvent has the following member function:
    a. rotINT32 GetEvent()

EnableRollover:
Use this function to enable a rollover for a touch region. It accepts the touch region's ID, two strings specifying the names of the on and off bitmap images, and a cache setting. The cache_image parameter simply tells the script whether to keep the bitmaps loaded in memory. This is a good idea except that it uses up memory very quickly so we recommend that cache_image is normally set to 0.

EnableRegion:
Use this function to enable or disable a rollover region. It accepts the touch region's ID and a Boolean value (true or false). The rollover regions default to enabled when created, but you can set up all of the regions at the start of your script and then just enable the current ones when required.

SetRollOverOrigin:
The default requirement is that rollover bitmaps are the same size and position as the touch region (though for circular regions the bitmap is obviously square). This function can be used to change the origin, so that more (or less) of the screen changes when the mouse rolls in and out of the region. This means that bitmaps which are larger than the region can be drawn. Note that the default origin for circular regions is (x - r, y - r) where (x, y) is the center and r is the radius.

ClearRegion:
Clears the list of regions added using AddRegion so that any touches in those regions no longer generate events. This call has no effect on the rollover graphics.

IsMousePresent:
Returns whether a relative pointing device is attached (i.e. not an absolute device like a touch screen).

Example: This code loops a video and waits for a mouse click or touch screen input. It outputs the coordinates, to the shell, of the click or touch, if it's within the defined region.

```
v=CreateObject("roVideoPlayer")
t=CreateObject("roTouchScreen")
p=CreateObject("roMessagePort")

v.SetPort(p)
t.SetPort(p)
v.SetLoopMode(1)
v.PlayFile("testclip.mp2v")

t.AddRectangleRegion(0,0,100,100,2)

loop:
    msg=wait(0, p)
    print "type: ";type(msg)
    print "msg=";msg
    if type(msg)="roTouchEvent" then
        print "x,y=";msg.GetX();msg.GetY()
    endif
    goto loop:
```

Another Example with Mouse support:

```
t=CreateObject("roTouchScreen")
```

```
t.SetPort(p)
REM Puts up a cursor if a mouse is attached
REM The cursor must be a 16 x 16 BMP
REM The x,y position is the “hot spot” point
t.SetCursorBitmap("cursor.bmp", 16, 16)
t.SetResolution(1024, 768)
t.SetCursorPosition(512, 389)
REM
REM Pass enable cursor display:  TRUE for on, and FALSE for off
REM The cursor will only enable if there is a mouse attached
REM
t.EnableCursor(TRUE)
```

Example with a Rollover Region and Mouse Support:

```
img=CreateObject("roImagePlayer")
t=CreateObject("roTouchScreen")
p=CreateObject("roMessagePort")
t.SetPort(p)

t.SetCursorBitmap("cursor.bmp", 16, 16)
t.SetResolution(1024, 768)
t.SetCursorPosition(512, 389)
t.EnableCursor(1)

img.DisplayFile("\menu.bmp")

REM Adds a rectangular touch region
REM Enables rollover support for that region
REM Sets the rollover origin to the same position as the touch region
REM
t.AddRectangleRegion(0, 0, 100, 100, 1)
t.EnableRollOver(1, "on.bmp", "off.bmp", true)
t.SetRollOverOrigin(1, 0, 0)
```

roSerialPort

This object controls the RS232 serial port, allowing you to receive input and send responses.

roSerialPort has these interfaces:

1. `ifStream`
2. `ifSerialControl`

ifStream has these member functions:

```
rotVOID SendByte(rotINT32 byte)
rotVOID SendLine(rotSTRING line)
rotVOID SendBlock(rotSTRING block)
rotVOID SetEol(rotSTRING eol)
rotVOID SetLineEventPort(rotOBJECT port)
rotVOID SetByteEventPort(rotOBJECT port)
```

ifSerialControl has these member functions:

```
rotBOOL SetBaudRate(roINT32 baud_rate)
```

Supported baud rates are:

```
1800,   2000,  2400,  3600,  4800,  7200, 9600,
12800,  14400,  19200,  23040,  28800,  38400,  57600,
115200
```

```
rotBOOL SetMode(rotSTRING mode)
```

- Set the serial mode in “8N1” syntax. The first character is the number of data bits and can either be 5, 6, 7 or 8. The second is the parity and can be “N”one, “O”dd or “E”ven. The third is the number of stop bits and can be 1 or 2.
- HD2000 only

```
rotBOOL SetEcho(rotBOOL enable)
```

- Enables or disables serial echo. Returns true on success and false on failure.
- HD2000 only

roSerialPort sends events of the following type:

1. `roStreamLineEvent` - The line event is generated whenever the end of line string set using SetEol is found and contains a rotString for the whole line.
2. `roStreamByteEvent` - The byte event is generated on every byte received.

Example: This code waits for a serial event, and echos the input received on the serial port to the shell

```
serial = CreateObject("roSerialPort", 0, 9600)
p = CreateObject("roMessagePort")
serial.SetLineEventPort(p)

serial_only:
msg = wait(0,p) ' Wait forever for a message.
if(type(msg) <> "roStreamLineEvent") goto serial_only 'Accept serial
messages only.
serial.SendLine(msg) ' Echo the message back to serial.
```

roDeviceInfo

The roDeviceInfo object implements the ifDeviceInfo interface only.

The ifDeviceInfo interface provides:

- `rotSTRING GetModel(rotVOID)`
  - Returns the model name for the BrightSign device running the script as a string. For example "HD600" or "HD2000".
- `rotSTRING GetVersion(rotVOID)`
  - Returns the version number of the BrightSign firmware running on the device. For example "1.3.14".
- `rotINT GetVersionNumber(rotVOID)`
  - Returns the version number of the BrightSign firmware running on the device in the more comparable numeric form of (major * 65536 + minor * 256 + build).
- `rotSTRING GetBootVersion(rotVOID)`
  - Returns the version number of the BrightSign boot firmware (also known as "safe mode") as a string. For example "1.0.4".
- `rotINT GetBootVersionNumber(rotVOID)`
  - Returns the version number of the BrightSign boot firmware (also known as "safe mode") in the more comparable numeric form of (major * 65536 + minor + 256 + build).
- `rotINT GetDeviceUptime(rotVOID)`
  - Returns the number of seconds that the device has been running since the last power cycle or reboot.
- `rotINT GetDeviceLifetime(rotVOID)`
  - Returns the estimated number of seconds that the device has been running since manufacture. The result is more accurate if the device is left switched on for long periods rather than being constantly switched on and off. This figure has only been updated since v1.1.x firmware.
- `rotINT GetDeviceBootCount(rotVOID)`
  - Returns the number of times the device has successfully booted since manufacture. This figure has only been updated since v1.1.x firmware.
- `rotSTRING GetDeviceUniqueId(rotVOID)`
  - Returns an identifier that if not an empty string is unique to the unit running the script.
  - On the HD2000, this string is the MAC address of the on board Ethernet.

Example:

```
di = CreateObject("roDeviceInfo")
print di.GetModel()
print di.GetVersion(), di.GetVersionNumber() print di.GetBootVersion(),
di.GetBootVersionNumber() print di.GetDeviceUptime(),
di.GetDeviceLifetime(), di.GetDeviceBootCount()

On a particular system generates:

HD2000
1.4.3          66563
1.0.4          65540
 478           581578         77
```

roRegistry (HD2000 only)

The registry is an area of memory where a small number of persistent settings can be stored. Access to the registry is available through the roRegistry object.

This object is created with no parameters.

- `CreateObject("roRegistry")`

The following methods are supported:

- `roList GetSectionList()`
    - returns a list with one entry for each registry section.
- `rotBOOL Delete(rotSTRING section)`
    - deletes the specified section and returns an indication of success.
- `rotBOOL Flush()`
    - flushes the registry out to persistent storage.

roRegistrySection (HD2000 only)

A section of the registry, enabling the organization of settings within the registry.

This object must be supplied with a "section" name on creation.

- `CreateObject("roRegistrySection", rotSTRING section)`

The roRegistrySection object implements the ifRegistrySection interface. This interface provides:

- `rotSTRING Read(rotSTRING key)`
    - reads and returns the value of the specified key.
- `rotBOOL Write(rotSTRING key, rotSTRING value)`
    - replaces the value of the specified key.
- `rotBOOL Delete(rotSTRING key)`
    - deletes the specified key.
- `rotBOOL Exists(rotSTRING key)`
    - returns true if the specified key exists.
- `rotBOOL Flush()`
    - flushes the contents of the registry out to persistent storage.
- `roList GetKeyList()`
    - returns a list containing one entry per registry key in this section.

Example:

```
registrySection = CreateObject("roRegistrySection", "widget-usage")
' An empty entry will read as the null string and therefore be
converted to zero.
hits = val(registrySection.Read("big-red-button-hits"))
hits = hits + 1
registrySection.Write("big-red-button-hits", strI(hits))
```

Writes do not always take immediate effect to prevent the system from exceeding the maximum number of writes on the I2C ROM. At most sixty seconds after a write to the registry it will automatically be written out to persistent storage. If for some reason the change must be written immediately then one of the flush functions should be called. Changes are automatically written prior to application exit.

roSystemTime (HD2000 only)

roSystemTime provides the ability to read and write the time stored in the RTC
This object supports getting and setting the time and time zone.

The roSystemTime object implements ifSystemTime. This interface provides:

- `ifDateTime GetLocalDateTime()`
- `ifDateTime GetUtcDateTime()`
- `ifDateTime GetZoneDateTime(rotSTRING timezone_name)`
- `rotBOOL SetLocalDateTime(roDateTime localDateTime)`
- `rotBOOL SetUtcDateTime(roDateTime utcDateTime)`
- `rotSTRING GetTimeZone()`
- `rotBOOL SetTimeZone(rotSTRING zone_name)`

Dates up to 1 January 2038 are supported.

The following are the supported time zones:
EST: US Eastern Time
CST: US Central Time
MST: US Mountain Time
PST: US Pacific Time
AKST: Alaska Time
HST: Hawaii-Aleutian Time with no Daylight Saving (Hawaii)
HST1: Hawaii-Aleutian Time with Daylight Saving
MST1: US MT without Daylight Saving Time (Arizona)
EST1: US ET without Daylight Saving Time (East Indiana)
AST: Atlantic Time
CST2: Mexico (Mexico City)
MST2: Mexico (Chihuahua)
PST2: Mexico (Tijuana)
BRT: Brazil Time (Sao Paulo)
NST: Newfoundland Time
AZOT: Azores Time
GMTBST: London/Dublin Time
WET: Western European Time
CET: Central European Time
EET: Eastern European Time
MSK: Moscow Time
SAMT: Delta Time Zone (Samara)
YEKT: Echo Time Zone (Yekaterinburg)
IST: Indian Standard Time
NPT: Nepal Time
OMST: Foxtrot Time Zone (Omsk)
JST: Japanese Standard Time
CXT: Christmas Island Time (Australia)
AWST: Australian Western Time
AWST1: Australian Western Time without Daylight Saving Time
ACST: CST, CDT, Central Standard Time, , Darwin, Australia/Darwin, Australian Central Time without Daylight Saving Time (Darwin)
AEST: Australian Eastern Time
AEST1: Australian Eastern Time without Daylight Saving Time (Brisbane)
NFT: Norfolk (Island) Time (Australia)
NZST: New Zealand Time (Auckland)
CHAST: , Fiji Time, , Fiji, Pacific/Fiji, Yankee Time Zone (Fiji)

SST: X-ray Time Zone (Pago Pago)
GMT: Greenwich Mean Time
GMT-1: 1 hour ahead of Greenwich Mean Time
GMT-2: 2 hours ahead of Greenwich Mean Time
GMT-3: 3 hours ahead of Greenwich Mean Time
GMT-4: 4 hours ahead of Greenwich Mean Time
GMT-5: 5 hours ahead of Greenwich Mean Time
GMT-6: 6 hours ahead of Greenwich Mean Time
GMT-7: 7 hours ahead of Greenwich Mean Time
GMT-8: 8 hours ahead of Greenwich Mean Time
GMT-9: 9 hours ahead of Greenwich Mean Time
GMT-10: 10 hours ahead of Greenwich Mean Time
GMT-11: 11 hours ahead of Greenwich Mean Time
GMT-12: 12 hours ahead of Greenwich Mean Time
GMT-13: 13 hours ahead of Greenwich Mean Time
GMT-14: 14 hours ahead of Greenwich Mean Time
GMT+1: 1 hour behind Greenwich Mean Time
GMT+2: 2 hours behind Greenwich Mean Time
GMT+3: 3 hours behind Greenwich Mean Time
GMT+4: 4 hours behind Greenwich Mean Time
GMT+5: 5 hours behind Greenwich Mean Time
GMT+6: 6 hours behind Greenwich Mean Time
GMT+7: 7 hours behind Greenwich Mean Time
GMT+8: 8 hours behind Greenwich Mean Time
GMT+9: 9 hours behind Greenwich Mean Time
GMT+10: 10 hours behind Greenwich Mean Time
GMT+11: 11 hours behind Greenwich Mean Time
GMT+12: 12 hours behind Greenwich Mean Time
GMT+13: 13 hours behind Greenwich Mean Time
GMT+14: 14 hours behind Greenwich Mean Time

roDateTime (HD2000 only)

roDateTime represents an instant in time.

The roDateTime object implements ifDateTime. This interface provides:

- `rotINT32 GetDayOfWeek()`
- `rotINT32 GetDay()`
- `rotINT32 GetMonth()`
- `rotINT32 GetYear()`
- `rotINT32 GetHour()`
- `rotINT32 GetMinute()`
- `rotINT32 GetSecond()`
- `rotINT32 GetMillisecond()`
- `rotVOID SetDay(rotINT32 day)`
- `rotVOID SetMonth(rotINT32 month)`
- `rotVOID SetYear(rotINT32 year)`
- `rotVOID SetHour(rotINT32 hour)`
- `rotVOID SetMinute(rotINT32 minute)`
- `rotVOID SetSecond(rotINT32 second)`
- `rotVOID SetMillisecond(rotINT32 millisecond)`
- `rotVOID AddSeconds(rotINT32 seconds)`

- `rotVOID SubtractSeconds(rotINT32 seconds)`
- `rotVOID AddMilliseconds(rotINT32 milliseconds)`
- `rotVOID SubtractMilliseconds(rotINT32 milliseconds)`
- `rotBOOL Normalize()`
    - Check that all the fields supplied are correct. It fails if values are out of bounds

A newly created object is at the time represented by zero seconds.
When used via the ifString interface ifDateTime will always use the sortable date format "YYYY-MM-DD hh:mm:ss".

roTimer (HD2000 only)

The roTimer object implements ifTimer and ifSetMessagePort. This ifTimer interface provides:

- `rotVOID SetTime(rotINT32 hour, rotINT32 minute, rotINT32 second, rotINT32 millisecond)`
- `rotVOID SetDate(rotINT32 year, rotINT32 month, rotINT32 day)`
- `rotVOID SetDayOfWeek(rotINT32 day_of_week)`
    - Set the time that you wish the event to trigger. In general if a value is -1 then it is a wildcard and will cause the event to trigger every time the rest of the specification matches. If there are no wildcards, then the timer will trigger only once, when the specified date/time occurs. It is possible using a combination of day and day_of_week to specify invalid combinations that will never occur.
    - If specifications include any wildcard then the second and millisecond specification must be zero. Events will be raised at most once a minute near the whole minute.
- `rotVOID SetDateTime(ifDateTime)`
    - Set the time that you wish the event to trigger from a roDateTime object. It is not possible to set wildcards using this method.
- `rotBOOL Start()`
    - Start the timer based on the current values specified via the above functions.
- `rotBOOL Stop()`
    - Stop the timer.

Example: This code creates a timer that triggers every 30 seconds.

```
st=CreateObject("roSystemTime")
timer=CreateObject("roTimer")
mp=CreateObject("roMessagePort")
timer.SetPort(mp)

timeout=st.GetLocalDateTime()

timeout.AddSeconds(30)
timer.SetDateTime(timeout)

timer.Start()

while true
      ev = wait(0, mp)
      if (type(ev) = "roTimerEvent") then
            print "timer event received"
            timeout=st.GetLocalDateTime()
            timeout.AddSeconds(30)
            timer.SetDateTime(timeout)
            timer.Start()
      else
            print "unexpected event received"
      endif
endwhile
```

Example: This code creates a timer that triggers every minute using wildcards in the timer spec.

```
st=CreateObject("roSystemTime")
timer=CreateObject("roTimer")
mp=CreateObject("roMessagePort")
timer.SetPort(mp)

timer.SetDate(-1, -1, -1)
timer.SetTime(-1, -1, 0, 0)
timer.Start()

while true
      ev = wait(0, mp)
      if (type(ev) = "roTimerEvent") then
            print "timer event received"
      else
            print "unexpected event received"
      endif
endwhile
```

Example: This code creates a timer that triggers once at a specific date / time.

```
timer=CreateObject("roTimer")
mp=CreateObject("roMessagePort")
timer.SetPort(mp)

timer.SetDate(2008, 11, 1)
timer.SetTime(0, 0, 0, 0)

timer.Start()

while true
      ev = wait(0, mp)
      if (type(ev) = "roTimerEvent") then
            print "timer event received"
      else
            print "unexpected event received"
      endif
endwhile
```

roReadFile, roCreateFile, roReadWriteFile, roAppendFile (HD2000only)

These objects provide file I/O functionality on the HD2000 using the ifStreamRead, ifStreamSend, ifStreamSeek, and ifFile interfaces.

Creating an roReadFile object opens the specified file for reading only. Object creation fails if the file does not exist. roReadFile implements ifStreamSeek and ifStreamRead.

- `CreateObject("roReadFile", rotSTRING filename)`

Creating an roCreateFile object opens an existing file or creates a new file. If the file exists, it is truncated to zero size. roCreateFile implements ifStreamSeek, ifStreamRead, ifStreamSend, and ifFile.

- `CreateObject("roCreateFile", rotSTRING filename)`

Creating an roReadWriteFile object opens an existing file for both reading and writing. Object creation fails if the file does not exist. The current position is set to the beginning of the file. roReadWriteFile implements ifStreamSeek, ifStreamRead, ifStreamSend, and ifFile.

- `CreateObject("roReadWriteFile", rotSTRING filename)`

Creating an roAppendFile object opens an existing file or creates a new file The current position is set to the end of the file and all writes are made to the end of the file. roAppendFile implements ifStreamSend, and ifFile.

- `CreateObject("roAppendFile", rotSTRING filename)`

The ifStreamRead interface provides:

- `rotVOID SetReceiveEol(rotSTRING eol_sequence)`
  - Set the EOL sequence when reading from the stream.
- `rotINT32 ReadByte()`
  - Reads a single byte from the stream, blocking if necessary. If the EOF is reached or there is an error condition, then a value less than 0 is returned.
- `rotINT32 ReadByteIfAvailable()`
  - Reads a single byte from the stream if one is available. If none is available, it returns immediately. A return value less than 0 indicates either that the EOF has been reached or no byte is available.
- `rotSTRING ReadLine()`
  - Reads until it finds a complete end of line sequence. If it fails to find the sequence within 4096 bytes, then it returns the 4096 bytes found. No data is discarded in this case.
- `rotSTRING ReadBlock(rotINT32 size)`
  - Reads the specified number of bytes. Size is limited to 65536 bytes. In the event of an EOF or an error, fewer bytes than requested will be returned. Any null bytes in the file will mask any further bytes.
- `rotBOOL AtEof()`
  - Returns true if an attempt has been made to read beyond the end of the file. If the current position is at the end of the file but no attempt has been made to read beyond it, this method will return false.

The ifStreamSend interface provides:

- `rotVOID SetSendEol(rotSTRING eol_sequence)`
  - Set the EOL sequence when writing to the stream.
- `rotVOID SendByte(rotINT32 byte)`
  - Writes the specified byte to the stream.
- `rotVOID SendLine(rotSTRING string)`
  - Writes the specified characters to the stream followed by the current EOL sequence.

- `rotVOID SendBlock(rotSTRING string)`
    - Writes the specified characters to the stream. Any null bytes will terminate the block.

The ifStreamSeek interface provides:

- `rotVOID SeekAbsolute(rotINT32 offset)`
    - Seeks to the specified offset. If the offset is beyond the end of the file, then the file will be extended upon the next write and any previously unoccupied space will be filled with null bytes.
- `rotVOID SeekRelative(rotINT32 offset)`
    - Seeks to the specified offset relative to the current position. If the ultimate offset is beyond the end of the file, then the file will be extended as described in SeekAbsolute.
- `rotVOID SeekToEnd()`
    - Seeks to the end of the file.
- `rotINT32 CurrentPosition()`
    - Retrieves the current position within the file.

The ifFile interface provides:

- `rotVOID Flush()`
    - Ensures that all writes have been written out to the file. This is done automatically when the object is destroyed (for example, by reassigning the variable containing it).

roTextField (HD2000 only)

A text field represents an area of the screen that can contain arbitrary text. This feature is intended for presenting diagnostic and usage information rather than for generating a user interface. The roTextField object implements the ifTextField and ifStreamSend interfaces.

The object is created with several parameters:

- `CreateObject("roTextField", rotINT xpos, rotINT ypos, rotINT width_in_chars, rotINT height_in_chars, rotOBJECT metadata)`
    - xpos = Horizontal coordinate for the top left of the text field.
    - ypos = Vertical coordinate for the top left of the text field. The top of the screen is zero.
    - width_in_chars = Width of the text field in character cells.
    - height_in_chars = Height of the text field in character cells.
    - metadata = Optionally a roAssociativeArray containing extra parameters for the text field. If you don't require this then pass zero.

Note that in TV modes a border around the screen may not be displayed due to overscanning. You may want to use the roVideoMode object’s GetSafeX and GetSafeY functions to ensure that the coordinates you use will be visible.

The metadata object supports the following extra parameters:

- "CharWidth" the width of each character cell in pixels.
- "CharHeight" the height of each character cell in pixels.
- "BackgroundColor" the background color of the text field as an integer specifying eight bits for each of red, green and blue in the form &Hrrggbb.
- "TextColor" the color of the text as an integer specifying eight bits for each of red, green and blue in the form &Hrrggbb.
- "Size" an alternative to "CharWidth" and "CharHeight" for specifying either normal size text (0) or double-sized text (1).

The ifTextField interface provides:

- `rotVOID Cls(rotVOID)`
    - Clear the text field.
- `rotINT GetWidth(rotVOID)`
    - Return the width of the text field.
- `rotINT GetHeight(rotVOID)`
    - Return the height of the text field.
- `rotVOID SetCursorPos(rotINT x, rotINT y)`
    - Move the cursor to the specified position. Subsequent output will appear at this position.
- `rotINT GetValue(rotVOID)`
    - Return the value of the character currently under the cursor.

The ifStreamSend interface provides (note – the ifStreamSend interface is also described in the section documenting the various Roku file objects – the interface is described again below in a manner more specific to the roTextField object):

- `rotVOID SendByte(rotINT byte)`
    - Write the character indicated by the specified number at the current cursor position within the text field and advance the cursor.
- `rotVOID SendLine(rotSTRING string)`
    - Write the characters specified at the current cursor position followed by the end of line sequence.
- `rotVOID SendBlock(rotSTRING string)`
    - Write the characters specified at the current cursor position and advance the cursor to one position beyond the last character.

- `rotVOID SetSendEol(rotSTRING string)`
    - Set the sequence sent at the end of a SendLine request. This should be left at the default value of chr(13) for normal use.

As with any object that implements the ifStreamSend interface, a text field can be written to using the PRINT #textfield syntax (see the example below).

It is also possible to write to a text field using the syntax PRINT #textfield, @pos where pos is the character position in the textfield. For example, if your textfield object has 8 columns and 3 rows, writing to position 17 writes to row 3, column 2 (positions 0-7 are in row 1; positions 8-15 are in row 2; positions 16-23 are in the last row).

When output reaches the bottom of the text field it will automatically scroll.

For a more complex roTextField example, download the file snake.bas from the software downloads section of www.rokulabs.com/BrightSign.

Example:

```
meta = CreateObject("roAssociativeArray")
meta.AddReplace("CharWidth", 20)
meta.AddReplace("CharHeight", 32)
meta.AddReplace("BackgroundColor", &H101010) ' Dark grey
meta.AddReplace("TextColor", &Hffff00) ' Yellow
vm = CreateObject("roVideoMode")
tf = CreateObject("roTextField", vm.GetSafeX(), vm.GetSafeY(), 20, 20,
meta)
print #tf, "Hello World"
tf.SetCursorPos(4, 10)
print #tf, "World Hello"
```

roAssociativeArray (HD2000 only)

An associative array (also knows as a map, dictionary or hash table) allows objects to be associated with string keys. The roAssociativeArray class implements the ifAssociativeArray interface.

This object is created with no parameters:

- `CreateObject("roAssociativeArray")`

The ifAssociativeArray interface provides:

- `rotVOID AddReplace(rotSTRING key, rotOBJECT value)`
    - Add a new entry to the array associating the supplied object with the supplied string. Only one object may be associated with a string so any existing object is discarded.
- `rotOBJECT Lookup(rotSTRING key)`
    - Look for an object in the array associated with the specified string. If there is no object associated with the string then an object implementing ifInt containing zero is returned.
- `rotBOOL DoesExist(rotSTRING key)`
    - Look for an object in the array associated with the specified string. If there is no associated object then false is returned. If there is such an object then true is returned.
- `rotBOOL Delete(rotSTRING key)`
    - Look for an object in the array associated with the specified string. If there is such an object then it is deleted and true is returned. If not then false is returned.
- `rotVOID Clear(rotVOID)`
    - Remove all objects from the associative array.

Example:

```
aa = CreateObject("roAssociativeArray")

aa.AddReplace("Bright", "Sign")
aa.AddReplace("TMOL", 42)

print aa.Lookup("TMOL")
print aa.Lookup("Bright")

Produces:

 42
Sign
```

roRectangle (HD2000 only)

This object is created with several parameters:

- `CreateObject("roRectangle", rotINT32 x, rotINT32 y, rotINT32 width, rotINT32 height)`

The interface provides:

- `rotVOID SetX(rotINT32 x)`
- `rotVOID SetY(rotINT32 y)`
- `rotVOID SetWidth(rotINT32 width)`
- `rotVOID SetHeight(rotINT32 height)`
- `roINT32 GetX(rotVOID)`
- `roINT32 GetY(rotVOID)`
- `roINT32 GetWidth(rotVOID)`
- `roINT32 GetHeight(rotVOID)`

SetRectangle calls honor the view mode/aspect ratio conversion mode setup by the user. If the user has set the videoplayer to letterbox the video, it will do so if the video doesn't fit exactly in the new rectangle.

roTextWidget (HD2000 only)

An object used for putting text on the screen.

Object creation:

- `CreateObject("roTextWidget", roRectangle r, rotINT32 line_count, rotINT32 text_mode, rotINT32 pause_time)`
  - r – roRectangle that contains the text
  - line_count – the number of lines of text to show in the rectangle
  - text_mode – 0 for an animated ‘teletype’ like view or 1 for static text.
  - pause_time – how long each string is displayed prior to displaying the next string

The object includes the following interfaces:

- `rotBOOL PushString(rotSTRING str)`
  - The string is added to the list of strings to display. Strings are displayed in order and when the end is reached it loops.
- `rotBOOL PopStrings(rotINT32 number_of_string_to_pop)`
  - Pops strings off the front of the list (last in first out). The popping is done the next time the widget wraps so that strings can be added and removed seamlessly to the widget.
- `rotINT32 GetStringCount()`
  - Returns the number of strings that will exist once any pending pops have taken place.
- `rotBOOL Clear()`
  - Clears the list of strings leaving the widget blank and ready for more PushString() calls.

This object also uses the ifWidget interface which provides:

- `rotBOOL SetForegroundColor(rotINT32 color)`
- `rotBOOL SetBackgroundColor(rotINT32 color)`
  - `color` is in ARGB format.
- `rotBOOL SetFont(rotSTRING font_filename)`
  - `font_filename` is a TrueType font, for example: `CF:/Ariel.ttf`
- `rotBOOL SetBackgroundBitmap(rotSTRING background_bitmap_filename, rotBOOL stretch)`
  - If `stretch` is true, the image is stretched to the size of the window.
- `rotBOOL SetSafeTextRegion(roRectangle rect)`
  - `rect` specifies the rectangle within the widget where the text can be drawn safely.
- `rotBOOL Show(rotVOID)`
  - Displays the widget – after creation, the widget is hidden until `Show()` is called.
- `rotBOOL Hide(rotVOID)`
  - Hides the widget.

The top 8 bits of the color value are ‘alpha’. Alpha has no effect for the foreground text color but does effect the widget background color. 0 is fully transparent and 255 is fully non-transparent. This feature allows ‘subtitle’ like effects. For example, a semi-transparent black box containing text over video.

roResourceManager (HD2000 only)

The roResourceManager is used for managing strings in multiple languages.

Object creation:

- `CreateObject("roResourceManager", rotSTRING filename)`
    - `filename` = the name of the file that contains all of the localized resource strings required by the user. This file must be in UTF-8 format.

The interface includes:

- `rotBOOL SetLanguage(rotSTRING language_identifier)`
    - Instructs the roResourceManager object to use the specified language. False is returned if there are no resources associated with the specified language.
- `rotSTRING GetResource(rotSTRING resource_identifier)`
    - Returns the resource string in the current language for a given resource identifier.

At present, the main use for the roResourceManager is for localizing the roClockWidget.

The resource file passed in on creation has the following format for each string entry:

```
[RESOURCE_IDENTIFIER_NAME_GOES_HERE]
eng "Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec"
ger "Jan|Feb|Mär|Apr|Mai|Jun|Jul|Aug|Sep|Okt|Nov|Dez"
spa "Ene|Feb|Mar|Abr|May|Jun|Jul|Ago|Sep|Oct|Nov|Dic"
fre "Jan|Fév|Mar|Avr|Mai|Jun|Jul|Aou|Sep|Oct|Nov|Déc"
ita "Gen|Feb|Mar|Apr|Mag|Giu|Lug|Ago|Set|Ott|Nov|Dic"
dut "Jan|Feb|Mar|Apr|Mei|Jun|Jul|Aug|Sep|Okt|Nov|Dec"
swe "Jan|Feb|Mar|Apr|Maj|Jun|Jul|Aug|Sep|Okt|Nov|Dec"
```

The name in the square brackets is the resource identifier. Each line after it is a language identifier followed by the resource string. Multiple roResourceManagers can be created.

A default “resources.txt” file is available from Roku’s website.

roClockWidget (HD2000 only)

roClockWidget puts a clock on the screen. It has no extra interface, only construction arguments. roClockWidget implements the ifTextWidget interface.

Object creation:

- `CreateObject("roClockWidget", roRectangle rect, roResourceManager res, rotINT32 display_type)`
  - `rect` = the rectangle in which the clock is displayed. Based on the size of the rectangle, the widget picks a font.
  - `display_type` = 0 for date only, 1 for clock only. To show both on the screen, two widgets must be created.

Example:

```
rect=CreateObject("roRectangle", 0, 0, 300, 60)
res=CreateObject("roResourceManager", "resources.txt")
c=CreateObject("roClockWidget", rect, res, 1)
c.Show()
```

The resource manager is passed in to the widget and the widget uses the following resources within resources.txt to display the time/date correctly.

Here are the 'eng' entries:

```
[CLOCK_DATE_FORMAT]
eng "%A, %B %e, %Y"
[CLOCK_TIME_FORMAT]
eng "%l:%M"
[CLOCK_TIME_AM]
eng "AM"
[CLOCK_TIME_PM]
eng "PM"
[CLOCK_DATE_SHORT_MONTH]
eng "Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec"
[CLOCK_DATE_LONG_MONTH]
eng
"January|February|March|April|May|June|July|August|September|October|No
vember|December"
[CLOCK_DATE_SHORT_DAY]
eng "Sun|Mon|Tue|Wed|Thu|Fri|Sat"
[CLOCK_DATE_LONG_DAY]
eng "Sunday|Monday|Tuesday|Wednesday|Thursday|Friday|Saturday"
```

The following are the control characters for the date/time format strings:

```
// Date format
//
// %a Abbreviated weekday name
// %A Long weekday name
// %b Abbreviated month name
// %B Full month name
// %d Day of the month as decimal 01 to 31
// %e Like %d, the day of the month as a decimal number, but without
leading zero
// %m Month name as a decimal 01 to 12
// %n Like %m, the month as a decimal number, but without leading zero
// %y Two digit year
// %Y Four digit year

// Time format
//
// %H The hour using 24-hour clock (00 to 23)
// %I The hour using 12-hour clock (01 to 12)
// %k The hour using 24-hour clock (0 to 23); single digits are
preceded by a blank.
// %l The hour using 12-hour clock (1 to 12); single digits are
preceded by a blank.
// %M Minutes (00 to 59)
// %S Seconds (00 to 59)
```

roUrlTransfer (HD2000 only)

This object is used for reading from and writing to remote servers through URLs.

This object is created with no parameters:

- `CreateObject("roUrlTransfer")`

The interface provides:

- rotINT32 `GetIdentity()`
  - Returns a magic number that can be used to identify whether events originated from this object.
- `rotBOOL SetUrl(rotSTRING URL)`
  - Sets the URL to use for the transfer request.
  - Returns false on failure – use GetFailureReason() to find out the reason for the failure.
- `rotVOID SetPort(ifMessagePort port)`
  - Set the message port to which events will be posted for asynchronous requests.
- `rotBOOL AddHeader(rotSTRING name, rotSTRING value)`
  - Add the specified HTTP header. Only valid for HTTP URLs.
  - Returns false on failure – use GetFailureReason() to find out the reason for the failure.
- `rotSTRING GetToString(rotVOID)`
  - Connect to the remote service as specified in the URL and return the response body as a string. This function cannot return until the exchange is complete and it may block for a long time.
  - Only having a single string return means that much of the information (headers, response codes) is discarded. If this information is required then use AsyncGetToString instead.
  - The size of string returned is limited to 65536 characters.
- `rotINT32 GetToFile(rotSTRING filename)`
  - Connect to the remote service as specified in the URL and write the response body to the specified file.
  - This function does not return until the exchange is complete and may block for a long time.
  - The response code from the server is returned. It is not possible to access any of the response headers. If this information is required use AsyncGetToFile instead.
- `rotBOOL AsyncGetToString(rotVOID)`
  - Begin a get request to a string asynchronously. Events will be sent to the message port associated with the object. If false is returned then the request could not be issued and no events will be delivered.
- `rotBOOL AsyncGetToFile(rotSTRING filename)`
  - Begin a get request to a file asynchronously. Events will be sent to the message port associated with the object. If false is returned then the request could not be issued and no events will be delivered.
- `roUrlEvent Head(rotVOID)`
  - Synchronously perform an HTTP HEAD request and return the resulting response code and headers through a roUrlEvent object. In the event of catastrophic failure (e.g. an asynchronous operation is already active) then a null object is returned.
- `rotBOOL AsyncHead(rotVOID)`
  - Begin an HTTP HEAD request asynchronously. Events will be sent to the message port associated with the object. If false is returned then the request could not be issued and no events will be delivered.
- `rotINT32 PostFromString(rotSTRING request)`
  - Use the HTTP POST method to post the supplied string to the current URL and return the response code. Any response body is discarded.
- `rotBOOL PostFromFile(rotSTRING filename)`
  - Use the HTTP POST method to post the contents of the file specified to the current URL and return the response code. Any response body is discarded.

- `rotBOOL AsyncPostFromString(rotSTRING request)`
    - Use the HTTP POST method to post the supplied string to the current URL. Events of type roUrlEvent will be sent to the message port associated with the object. If false is returned then the request could not be issued and no events will be delivered.
- `rotBOOL AsyncPostFromFile(rotSTRING filename)`
    - Use the HTTP POST method to post the contents of the specified file to the current URL. Events of type roUrlEvent will be sent to the message port associated with the object. If false is returned then the request could not be issued and no events will be delivered.
- `rotBOOL SetUserAndPassword(rotSTRING user, rotSTRING password)`
    - Enables HTTP authentication using the specified user name and password. Note that HTTP basic authentication is deliberately disabled due to it being inherently insecure. HTTP digest authentication is supported.
- `rotBOOL SetMinimumTransferRate(rotINT32 bytes_per_second, rotINT32 period_in_seconds)`
    - Cause the transfer to be terminated if the rate drops below `bytes_per_second` when averaged over `period_in_seconds`. Note that if the transfer is over the Internet you may not want to set `period_in_seconds` to a small number in case network problems cause temporary drops in performance. For large file transfers and a small `bytes_per_second` limit averaging over fifteen minutes or even longer might be appropriate.
- `rotSTRING GetFailureReason(rotVOID)`
    - If any of the roUrlTransfer functions indicate failure then this function may provide more information regarding the failure.

roUrlEvent (HD2000 only)

- `rotINT32 GetInt(rotVOID)`
    - Returns the type of event. The following event types are currently defined:
        - 1 – transfer complete
        - 2 – transfer started. Headers are available for suitable protocols. (Not currently implemented)
- `rotINT32 GetResponseCode(rotVOID)`
    - Returns the protocol response code associated with this event. For a successful HTTP request this will be the HTTP status code 200.
    - For unexpected errors the return value is negative. There are lots of possible negative errors from the CURL library but it's often best just to look at the text version via `GetFailureReason()`.

Here are some potential errors. Not all of them can be generated by BrightSign:

Status	Name	Description
-1	CURLE_UNSUPPORTED_PROTOCOL	
-2	CURLE_FAILED_INIT	
-3	CURLE_URL_MALFORMAT	
-5	CURLE_COULDNT_RESOLVE_PROXY	
-6	CURLE_COULDNT_RESOLVE_HOST	
-7	CURLE_COULDNT_CONNECT	
-8	CURLE_FTP_WEIRD_SERVER_REPLY	
-9	CURLE_REMOTE_ACCESS_DENIED	a service was denied by the server due to lack of access - when login fails this is not returned.
-11	CURLE_FTP_WEIRD_PASS_REPLY	
-13	CURLE_FTP_WEIRD_PASV_REPLY	
-14	CURLE_FTP_WEIRD_227_FORMAT	
-15	CURLE_FTP_CANT_GET_HOST	
-17	CURLE_FTP_COULDNT_SET_TYPE	
-18	CURLE_PARTIAL_FILE	
-19	CURLE_FTP_COULDNT_RETR_FILE	
-21	CURLE_QUOTE_ERROR	quote command failure
-22	CURLE_HTTP_RETURNED_ERROR	
-23	CURLE_WRITE_ERROR	
-25	CURLE_UPLOAD_FAILED	failed upload "command"
-26	CURLE_READ_ERROR	could open/read from file
-27	CURLE_OUT_OF_MEMORY	
-28	CURLE_OPERATION_TIMEDOUT	the timeout time was reached
-30	CURLE_FTP_PORT_FAILED	FTP PORT operation failed

-31	CURLE_FTP_COULDNT_USE_REST	the REST command failed
-33	CURLE_RANGE_ERROR	RANGE "command" didn't work
-34	CURLE_HTTP_POST_ERROR	
-35	CURLE_SSL_CONNECT_ERROR	wrong when connecting with SSL
-36	CURLE_BAD_DOWNLOAD_RESUME	couldn't resume download
-37	CURLE_FILE_COULDNT_READ_FILE	
-38	CURLE_LDAP_CANNOT_BIND	
-39	CURLE_LDAP_SEARCH_FAILED	
-41	CURLE_FUNCTION_NOT_FOUND	
-42	CURLE_ABORTED_BY_CALLBACK	
-43	CURLE_BAD_FUNCTION_ARGUMENT	
-45	CURLE_INTERFACE_FAILED	CURLOPT_INTERFACE failed
-47	CURLE_TOO_MANY_REDIRECTS	catch endless re-direct loops
-48	CURLE_UNKNOWN_TELNET_OPTION	User specified an unknown option
-49	CURLE_TELNET_OPTION_SYNTAX	Malformed telnet option
-51	CURLE_PEER_FAILED_VERIFICATION	peer's certificate or fingerprint wasn't verified fine
-52	CURLE_GOT_NOTHING	when this is a specific error
-53	CURLE_SSL_ENGINE_NOTFOUND	SSL crypto engine not found
-54	CURLE_SSL_ENGINE_SETFAILED	can not set SSL crypto engine as default
-55	CURLE_SEND_ERROR,	failed sending network data
-56	CURLE_RECV_ERROR	failure in receiving network data
-58	CURLE_SSL_CERTPROBLEM	problem with the local certificate
-59	CURLE_SSL_CIPHER	couldn't use specified cipher
-60	CURLE_SSL_CACERT	problem with the CA cert (path?)
-61	CURLE_BAD_CONTENT_ENCODING	Unrecognized transfer encoding
-62	CURLE_LDAP_INVALID_URL	Invalid LDAP URL
-63	CURLE_FILESIZE_EXCEEDED,	Maximum file size exceeded
-64	CURLE_USE_SSL_FAILED,	Requested FTP SSL level failed
-65	CURLE_SEND_FAIL_REWIND,	Sending the data requires a rewind that failed
-66	CURLE_SSL_ENGINE_INITFAILED	failed to initialise ENGINE
-67	CURLE_LOGIN_DENIED	user, password or similar was not accepted and we failed to login
-68	CURLE_TFTP_NOTFOUND	file not found on server
-69	CURLE_TFTP_PERM	permission problem on server
-70	CURLE_REMOTE_DISK_FULL	out of disk space on server
-71	CURLE_TFTP_ILLEGAL	Illegal TFTP operation
-72	CURLE_TFTP_UNKNOWNID	Unknown transfer ID
-73	CURLE_REMOTE_FILE_EXISTS	File already exists
-74	CURLE_TFTP_NOSUCHUSER	No such user

-75	CURLE_CONV_FAILED	conversion failed
-76	CURLE_CONV_REQD	caller must register conversion callbacks using curl_easy_setopt options CURLOPT_CONV_FROM_NETWORK_FUNCTION, CURLOPT_CONV_TO_NETWORK_FUNCTION, and CURLOPT_CONV_FROM_UTF8_FUNCTION
-77	CURLE_SSL_CACERT_BADFILE	could not load CACERT file, missing or wrong format
-78	CURLE_REMOTE_FILE_NOT_FOUND	remote file not found
-79	CURLE_SSH	error from the SSH layer, somewhat generic so the error message will be of interest when this has happened
-80	CURLE_SSL_SHUTDOWN_FAILED	Failed to shut down the SSL connection

- `rotSTRING GetFailureReason(rotVOID)`
  - Returns a description of the failure that occurred.
- `rotSTRING GetString(rotVOID)`
  - Return the string associated with the event. For transfer complete `AsyncGetToString`, `AsyncPostFromString` and `AsyncPostFromFile` requests this will be the actual response body from the server truncated to 65536 characters.
- `rotINT32 GetFailureReason(rotVOID)`
  - Returns a magic number that can be matched with the value returned by `roUrlTransfer.GetIdentity()` to determine where this event came from.
- `roAssociativeArray GetResponseHeaders(rotVOID)`
  - Returns an associative array containing all the headers returned by the server for appropriate protocols (such as HTTP).
- `roINT32 GetSourceIdentity()`
  - Returns a magic number that can be matched with the value returned by `roUrlTransfer.GetIdentity()` to determine where this event came from.

roRssParser, roRssArticle (HD2000 only)

roRssParser and roRssArticle class are used to display an RSS ticker on the screen.

roRssParser is created with no parameters:

- `CreateObject("roRssParser")`

The roRssParser interface provides:

- `rotBOOL ParseFile(rotSTRING filename)`
    - Parse an RSS feed from a file.
- `rotBOOL ParseString(rotSTRING filename)`
    - Parse an RSS feed from a string.
- `rotObject GetNextArticle(rotVOID)`
    - Get the next article parsed by the RSS parser. The articles are sorted in publication date order with the most recent article first. This returns an roRssArticle object if there is one, otherwise an rotINT32 is returned.

The roRssArticle interface provides:

- `rotSTRING GetTitle(rotVOID)`
    - The title of the RSS item.
- `rotSTRING GetDescription(rotVOID)`
    - The content of the RSS item.
- `rotSTRING GetDescription(rotVOID)`
    - Returns the difference in seconds for the publication date between this RSS item and the most recent item in the feed. This can be used by the user to decide that an article is too old to display.

Example:

```
u=CreateObject("roUrlTransfer")
u.SetUrl("http://www.lemonde.fr/rss/sequence/0,2-3208,1-0,0.xml")
u.GetToFile("tmp:/rss.xml")

r=CreateObject("roRssParser")
r.ParseFile("tmp:/rss.xml")

EnableZoneSupport(1)
b=CreateObject("roRectangle", 0, 668, 1024, 100)
t=CreateObject("roTextWidget", b, 3, 2, 2)
t.SetForegroundColor(&hD0D0D0)
t.Show()

article_loop:
a = r.GetNextArticle()
if type(a) = "roRssArticle" then
        t.PushString(a.GetDescription())
        goto article_loop
endif

loop:
sleep(1000)
goto article_loop
```

roNetworkConfiguration (HD2000 only)

Object creation:

- `CreateObject("roNetworkConfiguration", 0)`

The `ifNetworkConfiguration` interface provides (methods do not take effect until Apply is called):

- `rotVOID SetDHCP(rotSTRING key, rotOBJECT value)`
    - Enable DHCP. Disables all other settings.
- `rotBOOL SetIP4Address(rotSTRING ip)`
- `rotBOOL SetIP4Netmask(rotSTRING netmask)`
- `rotBOOL SetIP4Broadcast(rotSTRING broadcast)`
- `rotBOOL SetIP4Gateway(rotSTRING gateway)`
    - Set IPv4 interface configuration. All values must be set – no cleverness is applied (unlike ifconfig shell command). Parameter is a string dotted decimal quad (i.e. “192.168.1.2” or similar). Returns true on success.
    - Example
```
nc.SetIP4Address("192.168.1.42")
nc.SetIP4Netmask("255.255.255.0")
nc.SetIP4Broadcast("192.168.1.255")
nc.SetIP4Gateway("192.168.1.1")
```
- `rotBOOL SetDomain(rotSTRING domain)`
    - Set the device domain name. This will be appended to names to fully qualify them. It is not necessary to call this. Returns true on success.
    - Example
```
nc.SetDomain("roku.com")
```
- `rotVOID AddDNSServer(rotSTRING server)`
    - When the object is created there are no DNS servers, this adds another server to the list. There is currently a maximum of three but adding more will not fail. Returns true on success. There is no way to remove all the servers, just re-create the object.
- `rotSTRING GetFailureReason(rotVOID)`
    - Give more information when a member function has returned false.
- `rotBOOL Apply(rotVOID)`
    - Apply the requested changes to the network interface. This may take several seconds to complete.

roStorageInfo (HD2000 only)

Objects of this type are used to report storage device usage information.

Object creation:

- `CreateObject("roStorageInfo", rotSTRING path)`
    - Create a roStorageInfo object containing the storage device information for the specified path. The path need not be to the root of the storage device.

ifStorageInfo interface:

Note that on some filesystems that have a portion of space reserved for the super-user the

expression GetUsedInMegabytes() + GetFreeInMegabytes() == GetSizeInMegabytes() may not be true.

- `rotINT32 GetBytesPerBlock(rotVOID)`
    - Returns the size of a native block on the filesystem used by the storage device specified.
- `rotINT32` GetSizeInMegabytes`(rotVOID)`
    - Returns the total size of the storage device in Mibibytes.
- `rotINT32` GetUsedInMegabytes`(rotVOID)`
    - Returns the amount of space currently used on the storage device in Mibibytes. Note that this includes the size of the pool because this class does not know anything about pools.
- `rotINT32` GetFreeInMegabytes`(rotVOID)`
    - Returns the available space on the storage device in Mibibytes.

Example:

```
si=CreateObject(“roStorageInfo”, “CF:/”)
Print si.GetFreeInMegabytes(); “MiB free”
```

roBrightPackage (HD2000 only)

An roBrightPackage represents a zip file. The zip file can include arbitrary content or can be installed on a storage device to provide content and script updates (for example, to distribute updates via USB thumb drives).

Object creation:

- `CreateObject("roBrightPackage", rotSTRING filename)`
  - filename = The filename for the zip file

The interface provides:

- `rotVOID Unpack(rotSTRING path)`
  - path = The destination path for the extracted files, e.g. "ATA:/".
- `rotVOID SetPassword(rotSTRING password)`
  - password = The password specified when the zip file was created
  - roBrightPackage supports AES 128 and 256 bit encryption as generated by WinZip.

Example:

```
      package = CreateObject("roBrightPackage", "newfiles.zip")
      package.SetPassword("test")
      package.Unpack("ATA:/")
```

Using roBrightPackage to distribute new content:

The HD2000 checks storages for autorun scripts in the following order:

- External USB devices 1 through 9
- CF
- SD

In addition to looking for autorun.bas scripts, the HD2000 looks for autorun.zip files that contain a script name autozip.bas. If autozip.bas is encrypted, then the HD2000 uses the password stored in the registry in the section 'security' under the name 'autozipkey' to decrypt the file. If an autorun.zip file with an autozip.bas file is found and autozip.bas is decrypted, the HD2000 will execute the autozip.bas file. The autozip.bas file cannot reference any external files as it is the only file to be automatically uncompressed by the HD2000 prior to execution. The autozip.bas script unpacks the contents of the autorun.zip file to an installed storage device and reboots to complete the update.

Example:

```
' Content update application

r=CreateObject("roRectangle", 20, 668, 1240, 80)
t=CreateObject("roTextWidget",r,1,2,1)
r=CreateObject("roRectangle", 20, 20, 1200, 40)
t.SetSafeTextRegion(r)
t.SetForegroundColor(&hff303030)
t.SetBackgroundColor(&hffffffff)
t.PushString("Updating content from USB drive, please wait...")

package = CreateObject("roBrightPackage", "autorun.zip")
package.SetPassword("test")
package.Unpack("ATA:/")
package = 0

t.Clear()
```

```
t.PushString("Update complete - remove USB drive to restart.")

wait:
      sleep(1000)

      usb_key = CreateObject("roReadFile", "USB1:/autorun.zip")
      if type(usb_key) <> "roReadFile" then
            a=RebootSystem()
      endif
      usb_key = 0

      goto wait
```

roDatagramSender, roDatagramReceiver (HD2000 only)

The `roDatagramSender` and `roDatagramReceiver` classes allow for simple sending and receiving of unicast and broadcast UDP packets.

`roDatagramSender` allows UDP packets to be sent to a specified destination. It implements `ifDatagramSender`.

`roDatagramSender` is created with no parameters:

- `CreateObject("`**`roDatagramSender`**`")`

The `ifDatagramSender` interface provides:

- `rotBOOL SetDestination(rotSTRING destination_address, rotINT32 destination_port)`
    - Specify the destination IP address in dotted quad form along with the destination port. Returns true on success.
- `rotINT32 Send(rotSTRING packet)`
    - Send the specified data packet as a datagram. Returns zero on success or a negative error number on failure.

`roDatagramReceiver` causes events to be sent to a message port when UDP packets are received on the specified port. It implements `ifIdentity` and `ifSetMessagePort`.

`roDatagramReceiver` is created with a single parameter:

- `CreateObject("`**`roDatagramReceiver`**`", rotINT32 port)`
    - Specify the port on which to receive UDP packets.

Examples:

```
‘ This script broadcasts a single UDP packet containing “HELLO” to
‘ anyone on the network listening on port 21075.
sender = CreateObject(“roDatagramSender”)
sender.SetDestination(“255.255.255.255”, 21075)
sender.Send(“Hello”)


‘ This script listens for UDP packets on port 21075
receiver = CreateObject(“roDatagramReceiver”, 21075)
mp = CreateObject(“roMessagePort”)
receiver.SetPort(mp)
while true
      event = mp.WaitMessage(0)
      if type(event) = “roDatagramEvent” then
            print “Datagram: “; event
      endif
end while
```