Roku Object Reference

Matches HD600 Software Version: 1.1.28
Matches HD2000 Software Version: 1.1.20

Roku

California, USA
www.rokulabs.com

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.

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)
sw = CreateObject("roGpioControlPort")         'switch/led control port
sw.SetPort(p)

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

roClassMessagePort

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)`

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

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

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

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

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

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.Height()
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:

- Windows Media Player files for MPEG2 usually end in “mp2v” or “.mpg” to play in Windows XP. Note also that to play MPEG2 files in Windows you need to download and install an MPEG2 codec.
- Currently only MPEG2 files are supported by BrightSign
- Currently all video files must have a 48Khz audio track, even if it is silence.
- Currently only non-elementary MPEG2 video files are supported. The audio can be AC3 or PCM.

Interfaces:

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

- *ifAudioControl – see roAudioPlayer for docs*

- *ifVideoControl*
  ```
  rotBOOL SetViewMode(rotINT32 mode)
  ```

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

The view mode must be set before 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.

Note – SetViewMode is not currently supported on the HD2000

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.

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").
- Currently only .mp3 files supported

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)
  ```

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

  audio_ouput 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

  SetAudioOutput is not currently supported on the HD2000. The HD2000 implements 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

  SetAudioMode is not currently supported on the HD2000.

  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%
```

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.
2 Paused	Playback of the current media item is paused When a media item is paused, resuming playback begins from the same location.
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.

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)
  ```

roIRRemoteEvent

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);
```

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.

Note: DisplayFileEx and PreloadFileEx are not currently supported on the HD2000.

`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.

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.

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.

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
```

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)
```

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.

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 this member function:

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

Supported baud rates are:

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

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

This object returns the model and version.

This object has the following methods:

- `rotSTRING GetVersion()`
- `rotINT32 GetVersionNumber()`
- `rotSTRING GetModel()`

Example:

```
i = CreateObject("roDeviceInfo")
print i.GetModel()      ‘returns model number (HD600, HD2000, etc.)
HD600

print i.GetVersion()          ‘returns current version as a string
1.1.28

print i.GetVersionNumber()
65817
```

GetVersionNumber returns the current version as a number of the form
(major * 65536 + minor * 256 + build)