BrightSign®

OBJECT REFERENCE MANUAL

Firmware Version 4.4.x / 4.2.x /3.10.x

- BrightSign, LLC. 16795 Lark Ave., Suite 200 Los Gatos, CA 95032 | 408-852-9263 | www.brightsign.biz

TABLE OF CONTENTS

INTRODUCTION

We use BrightScript Objects (ROs) as the standardized way to expose functionality for our public SDKs. To publish a new API, we create a new BrightScript Object. All BrightSign players use this library of objects.

This Object Reference Manual describes the BrightScript Object architecture in two main sections:

- How to use BrightScript Objects (as a script writer)
- How the initial objects are defined for BrightSign players

INTERFACES AND METHODS OVERVIEW

Every BrightScript Object consists of one or more “interfaces.” An RO interface consists of one or more “methods.” For example, the *roVideoPlayer* object has two interfaces, *ifMediaTransport* and *ifSetMessagePort*. The interface *ifSetMessagePort* has one method, *SetPort*.

Example:

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

gpio = CreateObject("roControlPort", "BrightSign")
gpio.SetPort(p)
video.SetPort(p)
```

This syntax makes use of a shortcut 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: *The abstract interface* ifSetMessagePort *is exposed and implemented by both the* roControlPort *and the* roVideoPlayer *objects. Once the* SetPort *method is called, these objects will send their events to the supplied message port. This is discussed more in the Event Loops section below.*

BrightScript 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.

Classes

A "class name" is used to create a BrightScript Object. For example:

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

The class name of the above BrightScript Object is *roVideoPlayer*.

Object and Class Name Syntax

Class names have the following characteristics:

- Must start with an alphabetic character (a – z).
- May consist of alphabetic characters, numbers, or the "_" (i.e. underscore) symbol.
- Are not case sensitive.
- May be of any reasonable length.

Zones

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

A zone can contain video, images, audio, a clock, or text. There can be only one video zone per screen for all HD and AU models. However, there can be multiple zones of other types 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:

```
EnableZoneSupport(enable As Boolean) As Void
```

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.

Event Loops

When writing anything more than a very simple script, an "event loop" will need to be created. Event loops typically have this structure:

- o Wait for the event.
- o Process the event.
- o Return to step 1.

Events are things like a button that has been pressed, a timer that has been triggered, or a video that has finished playing back.

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

1. An object of the type *roMessagePort* is created in BrightScript by the user's script.
2. Objects 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 all the events go to this one port. To instruct the object to send events to a specific port, use the *ifSetMessagePort* interface.

3. The script waits for an event. The actual function to do this is *ifMessagePort.WaitMessage*, but if you are using BrightScript, the built-in statement “WAIT” allows you to do this easily.
4. If multiple event types are possible, your script should determine which event the wait received, then process it. The script then jumps back to the wait.

An event can be generated by any BrightScript Object. For example, the class *roControlPort* sends events of type *roControlDown* and *roControlUp*. The *roControlDown* implements the *ifInt* Interface, which allows access to an integer. An event loop needs to be aware of the possible events it can get and process them.

BrightSign Object Library

The following chapters specify each of the objects that can be used in BrightScript. A brief description, a list of interfaces, and the member functions of the interfaces are provided for each object class.

While most BrightScript objects have self-contained sections in this chapter, some objects are grouped in the same section if they are closely related or depend on one another to function correctly.

BRIGHTSCRIPT CORE OBJECTS

roArray

This object stores objects in a continuous array of memory locations. Since an *roArray* contains BrightScript components, and there are object wrappers for most intrinsic data types, entries can either be different types or all of the same type.

Object creation:

`CreateObject("roArray", *size* As Integer, *resize* As Boolean)`

- `size`: The initial number of elements allocated for an array.
- `resize`: If true, the array will be resized larger to accommodate more elements if needed. If the array is large, this process might take some time.
- `dim`: This statement may be used instead of CreateObject to create a new array. Dim is sometimes advantageous because it automatically creates array of array for multi-dimensional arrays.

Interfaces: *ifArray*, *ifEnum*, *ifArrayGet*, *ifArraySet*

The *ifArray* interface provides the following:

- `Peek() As Dynamic`: Returns the last (highest index) array entry without removing it.
- `Pop() As Dynamic`: Returns the last (highest index) entry and removes it from the array.
- `Push(a As Dynamic)`: Adds a new highest index entry to the end of the array
- `Shift() As Dynamic`: Removes index zero from the array and shifts all other entries down by one unit.
- `Unshift(a As Dynamic)`: Adds a new index zero to the array and shifts all other entries up by one unit.

- `Delete(a As Integer) As Boolean`: Deletes the indicated array entry and shifts all above entries down by one unit.
- `Count() As Integer` Returns the index of the highest entry in the array plus one (i.e. the length of the array).
- `Clear()`: Deletes every entry in the array.
- `Append(a As Object)`: Appends one *roArray* to another. If the passed *roArray* contains entries that were never set to a value, they are not appended.
  Note: *The two appended objects must be of the same type.*

The *ifEnum* interface provides the following:

- `Reset()`: Resets the position to the first element of enumeration.
- `Next() As Dynamic`: Returns a typed value at the current position and increment position.
- `IsNext() As Boolean`: Returns True if there is a next element.
- `IsEmpty() As Boolean`: Returns True if there is not an exact statement.

The *ifArrayGet* interface provides the following:

- `GetEntry(a As Integer) As Dynamic`: Returns an array entry of a given index. Entries start at zero. If an entry that has not been set is fetched, Invalid is returned.

The *ifArraySet* interface provides the following:

- `SetEntry(a As Integer, b As Dynamic)`: Sets an entry of a given index to the passed type value.

roAssociativeArray

An associative array (also known as a map, dictionary, or hash table) allows objects to be associated with string keys.

This object is created with no parameters:

```
CreateObject("roAssociativeArray")
```

Interfaces: *ifEnum*, *ifAssociativeArray*

The *ifEnum* interface provides the following:

- `Reset()`: Resets the position to the first element of enumeration.
- `Next() As Dynamic`: Returns the typed value at the current position and increment position.
- `IsNext() As Boolean`: Returns True if there is a next element.
- `IsEmpty() As Boolean`: Returns True if there is not a next element.

The *ifAssociativeArray* interface provides the following:

- `AddReplace(key As String, value As Object) As Void`: Adds 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.
- `Lookup(key As String) As Object`: Looks 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* and containing zero is returned.
- `DoesExist(key As String) As Boolean`: Looks 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.
- `Delete(key As String) As Boolean`: Looks 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.
- `Clear As Void`: Removes all objects from the associative array.

- `SetModeCaseSensitive():` Makes all subsequent actions case sensitive. All `roAssociativeArray` lookups are case insensitive by default.
- `LookupCi(a As String) As Dynamic`: Looks for an object in the array associated with the specified string. This method functions similarly to Lookup(), with the exception that key comparisons are always case insensitive, regardless of case mode.
- `Append(a As Object)`: Appends a second associative array to the first.

Example:

```
aa = CreateObject("roAssociativeArray")

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

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

The above script produces the following:

```
42
Sign
```

roBoolean

This is the object equivalent for the Boolean intrinsic type. It is useful in the following situations:

- When an object is needed instead of an intrinsic value. For example, if a Boolean is added to *roList*, it will be automatically wrapped in an *roBoolean* object by the language interpreter. When a function that expects a BrightScript component as a parameter is passed a Boolean, BrightScript automatically creates the equivalent BrightScript component.
- When any object exposes the *ifBoolean* interface. That object can then be used in any expression that expects an intrinsic value.

Interfaces: *ifBoolean*

The *ifBoolean* interface provides the following:

- `GetBoolean() As Boolean`
- `SetBoolean(a As Boolean)`

roByteArray

This object contains functions to convert strings to or from a byte array, as well as to or from ASCII hex or ASCII base 64. Note that if you are converting a byte array to a string, and the byte array contains a zero, the string conversion will end at that point. *roByteArray* will automatically resize to become larger as needed.

If you are converting a byte array to a string, and the byte array contains a zero, the string conversion will end at that point. *roByteArray* will autosize to become larger as needed. If you wish to disable this behavior, use the SetResize() function. If an uninitialized index is read, “invalid” is returned.

Since *roByteArray* supports the *ifArray* interface, it can be accessed with the `array []` operator. The byte array is always accessed as unsigned bytes while this interface is being used. This object also supports the *ifEnum* interface, and so can be used with a “`for each`” statement.

Interfaces: *ifByteArray*, *ifArray*, *ifArrayGet*, *ifEnum*, *ifArraySet*

See *roArray* for a description of *ifArray, ifArrayGet, ifEnum* and *ifArraySet*.

The *ifByteArray* interface provides the following:

- `WriteFile(filename As String) As Boolean`
- `WriteFile(filename As String, start_index As Integer, length As Integer) As Boolean`
- `ReadFile(filename As String) As Boolean`
- `ReadFile(filename As String, start_index As Integer, length As Integer) As Boolean`
- `AppendFile(filename As String) As Boolean`
- `SetResize(minimum_allocation_size As Integer, autoresize As Boolean)`
- `ToHexString() As String`
- `FromHexString(hexstring As String)`

- `ToBase64String() As String`
- `FromBase64String(base65string As String)`
- `ToAsciiString() As String`
- `FromAsciiString(a As String)`
- `GetSignedByte(index As Integer) As Integer`
- `GetSignedLong(index As Integer) As Integer`
- `IsLittleEndianCPU() As Boolean`

roDouble, roIntrinsicDouble

Interfaces: *ifDouble*

The *ifDouble* interface provides the following:

```
GetDouble() As Double
SetDouble(a As Double)
```

roFunction

Interfaces: *ifFunction*

- `GetSub() As Function`
- `SetSub(value As Function)`

roGlobal

Interfaces: *ifGlobal*

The *ifGlobal* interface provides the following:

- `Sleep(a As Integer)`
- `asc(a As String) As Integer`
- `chr(a As Integer) As String`
- `len(a As String) As Integer`
- `str(a As Double) As String`
- `strI(a As Integer) As String`
- `val(a As String) As Double`
- `abs(a As Double) As Double`
- `atn(a As Double) As Double`
- `cdbl(a As Integer) As Double`
- `cint(a As Double) As Integer`
- `cos(a As Double) As Double`
- `exp(a As Double) As Double`
- `fix(a As Double) As Integer`
- `int(a As Double) As Integer`
- `log(a As Double) As Double`
- `sgn(a As Double) As Integer`
- `sgnI(a As Integer) As Integer`
- `sin(a As Double) As Double`
- `sqr(a As Double) As Double`
- `tan(a As Double) As Double`

- `Left(a As String, b As Integer) As String`
- `Right(a As String, b As Integer) As String`
- `StringI(a As Integer, b As Integer) As String`
- `String(a As Integer, b As String) As String`
- `Mid(a As String, b As Integer, c As Integer) As String`
- `instr(a As Integer, b As String, c As String) As Integer`
- `GetInterface(a As Object, b As String) As Interface`
- `Wait(a As Integer, b As Object) As Object`
- `ReadAsciiFile(a As String) As String`
- `WriteAsciiFile(a As String, b As String) As Boolean`
- `ListDir(a As String) As Object`
- `MatchFiles(a As String, b As String) As Object`: Takes a directory to look in (it can be as simple as "." or "/") and a pattern to be matched and then returns an *roList* containing the results. The match is only applied in the specified directory; you will get no results if you have a pattern with a directory separator in it. The pattern is case insensitive.
- `LCase(a As String) As String`
- `UCase(a As String) As String`
- `DeleteFile(a As String) As Boolean`
- `DeleteDirectory(a As String) As Boolean`
- `CreateDirectory(a As String) As Boolean`
- `RebootSystem()`
- `ShutdownSystem()`
- `UpTime(a As Integer) As Float`
- `csng(a As Integer) As Float`
- `FormatDrive(a As String, b As String) As Boolean`: Formats the specified drive using one of the file systems listed below. This function returns True if successful and False in the event of failure:
  - `vfat` (DOS/Windows file system): Readable and writable by Windows, Linux, and MacOS.

    - `ext2` (Linux file system): Writable by Linux and readable by Windows and MacOS with additional software.
    - `ext3` (Linux file system): Writable by Linux and readable by Windows and MacOS with additional software. This file system uses journaling for additional reliability.
- `EjectDrive(a As String) As Boolean`
- `CopyFile(a As String, b As String) As Boolean`: Copies the specified source file-path name to the specified destination path name. The function returns True if successful and False in the event of failure.
- `MoveFile(a As String, b As String) As Boolean`: Moves the specified source file to the specified destination. The function returns True if successful and False in the event of failure.
  Note: *Both path names must be on the same drive.*
- `strtoi(a As String) As Integer`
- `rnd(a As Dynamic) As Dynamic`
- `RunGarbageCollector() As Object`
- `GetDefaultDrive() As String`: Returns the current default drive complete with a trailing slash. When running `autorun.brs`, the drive containing the autorun is designated as the current default.
- `SetDefaultDrive(a As String)`: Sets the current default drive, which does not need to include a trailing slash. This function will not fail. However, if the specified default drive is non-existent, it will not be possible to retrieve anything.
- `EnableZoneSupport(a As Boolean)`
- `EnableAudioMixer(a As Boolean)`
- `Pi() As Double`

roInt, roFloat, roString

The intrinsic types *roInt32*, *roFloat*, and *roString* have an object and interface equivalent. These are useful in the following situations:

- 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, an *roTouchEvent* can be used as an integer whose value is the *userid* of the *roTouchEvent*.

Note: *If "o" is an roInt, then these statements have the following effects*:

- o print o: Prints *o.GetInt()*
- o i%=o: Assigns the integer i% the value of *o.GetInt()*
- o k=o: Presumably k is *typeOmatic*, so it becomes another reference to the *roInt* o
- o o=5: This is NOT the same as o.SetInt(5). Instead it releases o, changes the type of o to *roINT32* (o is *typeOmatic*), and assigns it to 5.

When a function that expects a BrightScript Object as a parameter is passed an *int*, *float*, or *string*, BrightScript automatically creates the equivalent object.

Interfaces: *ifInt*, *ifIntOps*, *ifFloat*, *ifString*, *ifStringOps*

roInt contains the *ifInt* interface, which provides the following:

- `GetInt As Integer`
- `SetInt(value As Integer) As Void`

roInt also contains the *ifIntOps* interface, which provides the following:

- `ToStr() As String`

roFloat contains *the ifFloat* interface, which provides the following:

- `GetFloat As Float`
- `SetFloat(value As Float) As Void`

roString contains the *ifString* interface, which provides the following:

- `GetString As String`
- `SetString(value As String) As Void`

roString also contains the *ifStringOps* interface, which provides the following:

- `SetString(a As String, b As Integer)`
- `AppendString(a As String, b As Integer)`
- `Len() As Integer`
- `GetEntityEncode() As String`
- `Tokenize(a As String) As Object`
- `Trim() As String`
- `ToInt() As Integer`
- `ToFloat() As Float`
- `Left(a As Integer) As String`
- `Right(a As Integer) As String`
- `Mid(a As Integer) As String`
- `Mid(a As Integer, b As Integer) As String`
- `Instr(a As String) As Integer`
- `Instr(a As Integer, b As String) As Integer`

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())
```

An integer value of "5" is converted to type *roInt* automatically because *list.AddTail* expects a BrightScript Object as its parameter.

Example: Here the function *ListDir* returns an object *roList* of *roStrings*:

```
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
```

roList

This object functions as a general-purpose, doubly linked list. It can be used as a container for arbitrary-length lists of BrightSign Objects. The array operator `[ ]` can be used to access any element in an ordered list.

Interfaces: *ifList*, *ifEnum*, *ifArray*, *ifArrayGet*, *ifArraySet*

The *ifList* interface provides the following:

- `Count() As Integer`: Returns the number of elements in the list.
- `ResetIndex() As Boolean:` Resets the current index or position in the list to the head element.
- `AddTail(obj As Object) As Void`: Adds a typed value to the tail of the list.
- `AddHead(obj As Object) As Void`: Adds a typed value to the head of the list.
- `RemoveIndex As Object`: Removes an entry from the list at the current index or position and increments the index or position in the list. It returns Invalid when the end of the list is reached.
- `GetIndex As Object`: Retrieves an entry from the list at the current index or position and increments the index or position in the list. It returns Invalid when the end of the list is reached.
- `RemoveTail As Object`: Removes the entry at the tail of the list.
- `RemoveHead As Object`: Removes the entry at the head of the list.
- `GetTail As Object`: Retrieves the entry at the tail of the list and keeps the entry in the list.
- `GetHead As Object`: Retrieves the entry at the head of the list and keeps the entry in the list.
- `Clear()`: Removes all elements from the list.

The *ifEnum* interface provides the following:

- `Reset()`: Resets the position to the first element of enumeration.
- `Next() As Dynamic`: Returns the typed value at the current position and increment position.
- `IsNext() As Boolean`: Returns True if there is a next element.
- `IsEmpty() As Boolean`: Returns True if there is not a next element.

The *ifArray* interface provides the following:

- `Peek() As Dynamic`: Returns the last (highest index) array entry without removing it.
- `Pop() As Dynamic`: Returns the last (highest index) entry and removes it from the array.
- `Push(a As Dynamic)`: Adds a new highest index entry to the end of the array
- `Shift() As Dynamic`: Removes index zero from the array and shifts all other entries down by one unit.
- `Unshift(a As Dynamic)`: Adds a new index zero to the array and shifts all other entries up by one unit.
- `Delete(a As Integer) As Boolean`: Deletes the indicated array entry and shifts all above entries down by one unit.
- `Count() As Integer` Returns the index of the highest entry in the array plus one (i.e. the length of the array).
- `Clear()`: Deletes every entry in the array.
- `Append(a As Object)`: Appends one *roArray* to another. If the passed *roArray* contains entries that were never set to a value, they are not appended.
  Note: *The two appended objects must be of the same type.*

The *ifArrayGet* interface provides the following:

- `GetEntry(a As Integer) As Dynamic`: Returns an array entry of a given index. Entries start at zero. If an entry that has not been set is fetched, Invalid is returned.

The *ifArraySet* interface provides the following:

- `SetEntry(a As Integer, b As Dynamic)`: Sets an entry of a given index to the passed type value.

roRegex

This object allows the implementation of the regular-expression processing provided by the PCRE library.

This object is created with a string to represent the “matching-pattern” and a string to indicate flags that modify the behavior of one or more matching operations:

```
CreateObject(“roRegex”, “[a-z]+”, “i”)
```

The match string (in the example above, `“[a-z]+”`, which matches all lowercase letters) can include most Perl compatible regular expressions found in the PCRE documentation.

This object supports any combination of the following behavior flags (in the example above, `“i”`, which can be modified to match both uppercase and lowercase letters):

- `“i”`: Case-insensitive match mode.
- `“m”`: Multiline mode. The start-line (“^”) and end-line (“$”) constructs match immediately before or after any `newline` in the subject string. They also match at the absolute beginning or end of a string.
- `“s”`: Dot-all mode, which includes a newline in the “`.*`” regular expression. This modifier is equivalent to “`/s`” in Perl.
- `“x”`: Extended mode, which ignores whitespace characters except when escaped or inside a character class. This modifier is equivalent to “`/x`” in Perl.

Interfaces: *ifRegex*

The *ifRegex* interface provides the following:

- `IsMatch(a As String) As Boolean`: Returns True if the string is consistent with the matching pattern.

- `Match(a As String) As Object`: Returns an *roArray* of matched substrings from the string. The entire match is returned in the form `array[0]`.This will be the only entry in the array if there are no parenthetical substrings. If the matching pattern contains parenthetical substrings, the relevant substrings will be returned as an array of length `n+1`, where `array[0]` is the entire match and each additional entry in the array is the match for the corresponding parenthetical expression.
- `Replace(a As String, b As String) As String`: Replaces the first occurrence of a match to the matching pattern in the string with the subset. The subset may contain numbered back-references to parenthetical substrings.
- `ReplaceAll(a As String, b As String) As String`: Performs a global search and replace.
- `Split(a As String) As Object`: Uses the matching pattern as a delimiter and splits the string on the delimiter boundaries. The function returns an *roList* of strings that were separated by the matching pattern in the original string.

roXMLElement

This object is used to contain an XML tree.

The *roXMLElement* object is created with no parameters:

```
CreateObject(“roXMLElement”)
```

The following examples illustrate how XML elements are parsed in BrightScript:

```
<tag1>This is example text</tag1>
```

Name = tag1
Attributes = Invalid
Body = *roString* containing “This is example text”

```
<tag2 caveman=”barney”/>
```

Name = tag2
Attributes = *roAssociativeArray* with one entry, {caveman, barney}
Body = Invalid

If the tag contains other tags, the body will be of the type *roXMLList*.

To generate XML content, create an *roXMLElement* and call the `SetBody()` and `SetName()` methods to build it. Then call the `GenXML()` method to generate it. See the following example:

```
root.SetName("myroot")
root.AddAttribute("key1","value1")
root.AddAttribute("key2","value2")
ne=root.AddBodyElement()
ne.SetName("sub")
ne.SetBody("this is the sub1 text")
ne=root.AddBodyElement()
ne.SetName("subelement2")
ne.SetBody("more sub text")
ne.AddAttribute("k","v")
ne=root.AddElement("subelement3")
ne.SetBody("more sub text 3")
root.AddElementWithBody("sub","another sub (#4)")
PrintXML(root, 0)
print root.GenXML(false)
```

Interfaces: *ifXMLElement*

The *ifXMLElement* interface provides the following:

- `GetBody() As Object`
- `GetAttributes() As Object`
- `GetName() As String`
- `GetText() As String`
- `GetChildElements() As Object`
- `GetNamedElements(a As String) As Object`
- `GetNamedElementsCi(a As String) As Object`

- `SetBody(a As Object)`: Generates an roXMLList for the body if needed. The method then adds the passed item (which should be an roXMLElement tag).
- `AddBodyElement() As Object`
- `AddElement(a As String) As Object`
- `AddElementWithBody(a As String, b As Object) As Object`
- `AddAttribute(a As String, b As String)`
- `SetName(a As String)`
- `Parse(a As String) As Boolean`
- `GenXML(a As Object) As String`: Generates XML content. This method takes a single Boolean parameter, indicating whether or not the XML should have an `<?xml` …> tag at the top.
- `Clear()`
- `GenXMLHdr(a As String) As String`
- `IsName(a As String) As Boolean`
- `HasAttribute(a As String) As Boolean`
- `ParseFile(a As String) As Boolean`

The following is an example subroutine to print out the contents of an *roXMLElement* tree:

```
PrintXML(root, 0)

Sub PrintXML(element As Object, depth As Integer)
    print tab(depth*3);"Name: ";element.GetName()
    if not element.GetAttributes().IsEmpty() then
        print tab(depth*3);"Attributes: ";
        for each a in element.GetAttributes()
            print a;"=";left(element.GetAttributes()[a], 20);
            if element.GetAttributes().IsNext() then print ", ";
        end for
        print
    end if
    if element.GetText()<>invalid then
        print tab(depth*3);"Contains Text: ";left(element.GetText(), 40)
    end if
    if element.GetChildElements()<>invalid
        print tab(depth*3);"Contains roXMLList:"
        for each e in element.GetChildElements()
            PrintXML(e, depth+1)
        end for
    end if
    print
end sub
```

roXMLList

Interfaces: *ifList*, *ifEnum*, *ifArray*, *ifArrayGet*, *ifArraySet*, *ifXMLList*

The *ifList* interface provides the following:

- `GetHead() As Dynamic`: Retrieves the entry at the head of the list and keeps the entry in the list.
- `GetTail() As Dynamic`: Retrieves the entry at the tail of the list and keeps the entry in the list.
- `RemoveHead() As Dynamic`: Removes the entry at the head of the list.
- `RemoveTail() As Dynamic`: Removes the entry at the tail of the list.
- `GetIndex() As Dynamic`: Retrieves an entry from the list at the current index or position and increments the index or position in the list. It returns Invalid when the end of the list is reached.
- `RemoveIndex() As Dynamic`: Removes an entry from the list at the current index or position and increments the index or position in the list. It returns Invalid when the end of the list is reached.
- `AddHead(a As Dynamic)`: Adds a typed value to the head of the list.
- `AddTail(a As Dynamic)`: Adds a typed value to the tail of the list.
- `ResetIndex() As Boolean`: Resets the current index or position in the list to the head element.
- `Count() As Integer`: Returns the number of elements in the list.
- `Clear()`: Removes all elements from the list.

The *ifEnum* interface provides the following:

- `Reset()`: Resets the position to the first element of enumeration.
- `Next() As Dynamic`: Returns the typed value at the current position and increment position.
- `IsNext() As Boolean`: Returns True if there is a next element.
- `IsEmpty() As Boolean`: Returns True if there is not a next element.

The *ifArray* interface provides the following:

- `Peek() As Dynamic`: Returns the last (highest index) array entry without removing it.
- `Pop() As Dynamic`: Returns the last (highest index) entry and removes it from the array.
- `Push(a As Dynamic)`: Adds a new highest index entry to the end of the array
- `Shift() As Dynamic`: Removes index zero from the array and shifts all other entries down by one unit.
- `Unshift(a As Dynamic)`: Adds a new index zero to the array and shifts all other entries up by one unit.
- `Delete(a As Integer) As Boolean`: Deletes the indicated array entry and shifts all above entries down by one unit.
- `Count() As Integer` Returns the index of the highest entry in the array plus one (i.e. the length of the array).
- `Clear()`: Deletes every entry in the array.
- `Append(a As Object)`: Appends one *roArray* to another. If the passed *roArray* contains entries that were never set to a value, they are not appended.
  Note: *The two appended objects must be of the same type.*

The *ifArrayGet* interface provides the following:

- `GetEntry(a As Integer) As Dynamic`: Returns an array entry of a given index. Entries start at zero. If an entry that has not been set is fetched, Invalid is returned.

The *ifArraySet* interface provides the following:

- `SetEntry(a As Integer, b As Dynamic)`: Sets an entry of a given index to the passed type value.

The *ifXMLList* interface provides the following:

- `GetAttributes() As Object`
- `GetText() As String`
- `GetChildElements() As Object`

- `GetNamedElements(a As String) As Object`: Returns a new XMLList that contains all *roXmlElements* that match the name of the passed element. This action is the same as using the dot operator on an *roXmlList*.
- `GetNamedElementsCi(a As String) As Object`
- `Simplify() As Object`: Returns an *roXmlElement* if the list contains exactly one element. Otherwise, it will return itself.

roAudioEventMx

Interfaces: *ifInt*, *ifSourceIdentity*, *ifAudioUserData*

The *ifInt* interface provides the following:

- `GetInt() As Integer`
- `SetInt(a As Integer)`

The *ifSourceIdentity* interface provides the following:

- `GetSourceIdentity() As Integer`
- `SetSourceIdentity() As Integer`

The *ifAudioUserData* interface provides the following:

- `GetSourceIdentity() As Integer`
- `SetSourceIdentity() As Integer`

roAudioOutput

This object allows individual control of audio outputs on the player.

Object Creation:

```
CreateObject("roAudioOutput", output As String)
```

The audio output parameter can take the following strings:

- `Analog`
- `SPDIF`
- `HDMI`
- `USB`
- `NONE`

These strings can be extended if future BrightSign players have multiple channels of the same type of audio output. For example, `Analog` could be extended to `Analog:1` or `Analog:0-2`.

You can create any number of *roAudioOutput* objects. There can be multiple instances of this object that represent the same audio output, but in these cases one object will override another.

Interfaces: *ifAudioOutput*

The *ifAudioOuput* interface provides the following:

- `SetVolume(a As Integer) As Boolean`: Sets the volume of the specified output as a percentage represented by an integer between 0 and 100.
- `SetMute(a As Boolean) As Boolean`: Mutes the specified output if True. This method is set to False by default.

- `GetOutput() As String`: Returns the string with which the *roAudioOutput* object was created.

The `SetVolume` and `SetMute` methods work in conjunction with the volume and mute functionality offered by *roAudioPlayer*. The *roAudioPlayer* volume settings affect the audio decoder volume. The audio stream is then sent to the assigned outputs, which have an additional level of volume control enabled by *roAudioOutput*.

Note: *To control which audio outputs connect to audio player outputs generated by* roAudioOutput*, use the* `SetPcmAudioOutputs` *and* `SetCompressedAudioOutputs` *methods, which can be used for* roVideoPlayer *and* roAudioPlayer*. See the* roAudioPlayer *entry for further explanation of these methods.*

The *roAudioOutput* object affects the absolute volume (as well as mute settings) for an audio output. If two players are streaming to the same output, both will be affected by any settings implemented through *roAudioOutput*.

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 the type *roAudioEvent*. All object calls are asynchronous. In other words, audio playback is handled in a different thread from the script. The script may continue to run while audio is playing.

Interfaces: *ifIdentity*, *ifSetMessagePort*, *ifMediaTransport*, *ifAudioControl*.

The *ifIdentity* interface provides the following:

- `GetIdentity() As Integer`

The *ifSetMessagePort* interface provides the following:

- `SetPort(As Object) As Void`
- `SetPort(a As Object)`

See *roVideoPlayer* for a description of *ifMediaTransport*.

The *ifAudioControl* interface provides the following:

- `SetPcmAudioOutputs(a As Object) As Boolean`: Determines which PCM audio outputs are connected to audio player outputs generated by *roAudioOutput*. This method takes as its argument one or more outputs in the form of an *roArray* of *roAudioOutput* parameters.
- `SetCompressedAudioOutputs(a As Object) As Boolean`: Determines which compressed audio outputs are connected to audio player outputs generated by *roAudioOutput*. This method takes as its argument one or more outputs in the form of an *roArray* of *roAudioOutput* parameters.

Note: *When one or both of these output methods are called, they will override the settings of the following* ifAudioControl *methods:*

- o `SetAudioOutput`

  - `MapStereoOutput`
  - `SetUsbAudioPort`
  - `MapDigitalOutput`
- `SetAudioOutput(audio_output As Integer) As Boolean`
- `SetAudioMode(audio_mode As Integer) As Boolean`
- `MapStereoOutput(mapping As Integer) As Boolean`
- `MapDigitalOutput(mapping As Integer) As Boolean`

Note: `MapDigitalOutput` *is not available on the HD2000*.

- `SetVolume(volume As Integer) As Boolean`
- `SetChannelVolumes(channel_mask As Integer, volume As Integer) As Boolean`
- `SetUsbAudioPort(a As Integer) As Boolean`
- `SetSpdifMute(a As Boolean) As Boolean`
- `StoreEncryptionKey(a As String, b As String) As Boolean`
- `StoreObfuscatedEncryptionKey(a As String, b As String) As Boolean`
- `SetStereoMappingSpan(a As Integer) As Boolean`
- `ConfigureAudioResources() As Boolean`
- `SetAudioStream(stream_index As Integer) As Boolean`

Note: *The following "Aux" functions are implemented only on the HD2000*

- `SetAudioOutputAux(audio_output As Integer) As Boolean`
- `SetAudioModeAux(audio_mode As Integer) As Boolean`
- `MapStereoOutputAux(mapping As Integer) As BooleanSetVolumeAux(volume As Integer) As Boolean`
- `SetChannelVolumesAux(channel_mask As Integer, volume As Integer) As Boolean`
- `SetAudioStreamAux(stream_index As Integer) As Boolean`

A call to *video.Stop* is needed before changing the audio output when a video file is playing or has played.

Example: This example shows how to use the `SetPcmAudioOutputs` and `SetCompressedAudioOutputs` methods in conjunction with *roAudioOutput*.

```
ao1=CreateObject("roAudioOutput", "Analog")
ao2=CreateObject("roAudioOutput", "HDMI")
ao3=CreateObject("roAudioOutput", "SPDIF")

v1=CreateObject("roVideoPlayer")
v1.SetPcmAudioOutputs(ao1)
```

--or--

```
ar = CreateObject("roArray", 2, true)
ar[0] = ao2
ar[1] = ao3
v1.SetCompressedAudioOutputs(ar)
```

Now the video player has been configured to output decoded audio to the analog output or compressed audio to the HDMI and SPDIF outputs.

audio_output values

0 – Analog audio
1 – USB audio
2 – Digital audio, stereo PCM
3 – Digital audio, raw AC3
4 – Onboard analog audio with HDMI mirroring raw AC3

digital audio values
0 – Onboard HDMI
1 – SPDIF from expansion module

audio_mode values: Options 0 and 1 only apply to video files; while 2 applies to all audio sources.
0 – AC3 Surround
1 – AC3 mixed down to stereo
2 – No audio
3 – Left
4 – Right

mapping values: Used to select which analog output to use if audio_output is set to 0.
0 – Stereo audio is mapped to onboard analog output
1 – Stereo audio is mapped to expansion module leftmost output
2 – Stereo audio is mapped to expansion module middle output
3 – Stereo audio is mapped to expansion module rightmost output

set_volume: Volume functions as a percentage and therefore takes a value between 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.
Note: *Separate volume levels are stored for roAudioPlayer and roVideoPlayer.*

set_channel_volumes: You can control the volume of individual audio channels. This volume command takes a hex channel mask, which determines the channels to apply the volume to, and a level, which is a percentage of the full scale. The volume control works according to audio channel rather than the output. The channel mask is a bit mask with the following bits for MP3 output:

- &H01 Left

- &H02 Right
- &H03 Both left and right

Example: This code sets audio output to the rightmost expansion moduleaudio port.

```
video = CreateObject("roVideoPlayer")

video.SetAudioOutput(0)
video.MapStereoOutput(3)
```

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(&H03, 65)     ‘all channels to 65%
```

Playing Multiple Audio Files Simultaneously

Multiple MP3 files, as well as the audio track of a video file, can be played to any combination of the following:

- Analog outputs
- 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 the *ifMediaTransport* `Stop` or `StopClear` calls).

Notes on multiple audio-file functionality:

- The onboard analog audio output and HDMI output are clocked by the same sample-rate clock. Therefore, if different content is being played out of each, the content must have the same sample rate.
- Currently, only a single set of USB speakers is supported.
- 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 determine whether a given set of audio files can be played at the same time as a video. The maximum recommended usage is a 16Mbps video file with three simultaneous MP3 160kbps streams.

Example: This code plays a video with audio over HDMI and an MP3 file to the onboard analog port.

```
video=CreateObject("roVideoPlayer")

video.SetAudioOutput(3)
video.PlayFile("video.mpg")

audio=CreateObject("roAudioPlayer")

audio.MapStereoOutput(0)
audio.PlayFile("audio.mp3")
```

roAudioPlayerMx

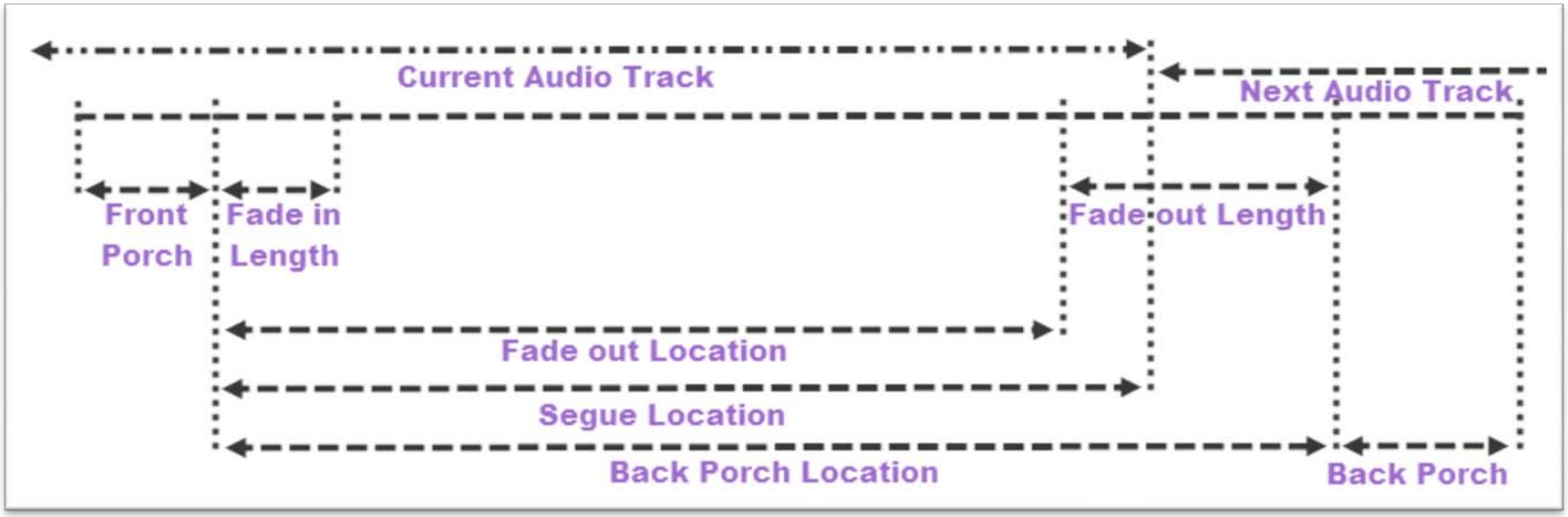
This object allows you to mix audio files. Each `roAudioPlayerMx` object conatins two internal audio players: The main audio playlist consists of queued audio tracks that play sequentially, while the audio overlay plays files on top of the main playlist. A fade will not occur if it is called while an overlay is playing, but the next audio track will start playing as expected.

Tracks are queued to `PlayFile` with their fade parameters specified in an associative array. These are the parameters you can pass to `PlayFile`:

- **Filename**: The filename of the track
- **FrontPorch**: The length, in milliseconds (ms), to skip from the start of the track. This value is 0 by default.
- **FadeOutLocation**: The location, in milliseconds (ms), of the fade out relative to the value of the **FrontPorch**. If the value is 0 (which is the default setting), and the **FadeOutLength** has a non-zero value, then the fade out is calculated back from the end of the file.
- **FadeOutLength**: The length of the fade out in milliseconds (ms). This value is 0 by default.
- **SegueLocation**: The location, in milliseconds (ms), of the event that triggers the next audio file to play. This location is relative to the first audio file that is played. If the **SegueLocation** parameter is not passed to `PlayFile`, the value defaults to the **FadeOutLocation**.
- **BackPorchLocation**: The location, in milliseconds (ms), of the termination point for the audio track. This location is relative to the first audio file that is played. If the **BackPorchLocation** parameter is not passed to `PlayFile`, the audio file plays to the end. The value is 0 by default, which disables the back porch.
- **TrackVolume**: The relative volume of of the audio track, measured as a percentage. Specify the percentage using values between 0 and 100.
- **EventID**: The ID for an audio event
- **EventTimeStamp**: The timestamp for the audio event. There can only be one event per audio file.
- **QueueNext**: The queuing of an audio track. Set the parameter value to 1 to queue an audio file to play after the current track.

- **Overylay**: The overlay specification of an audio track. Set the parameter value to 1 to fade down the main audio playlist while playing the audio track as an overaly. Overlays have additional parameters:
  - **AudioBedLevel**: The volume-level percentage of the main audio playlist while the overlay is playing. Specify the percentage using values between 0 and 100.
  - **AudioBedFadeOutLength**: The fade-out length of the main audio playlist.
  - **AudioBedFadeInLength**: The fade-in lenth for the length of the underlying audio track once the segure is triggered.
- **FadeCurrentPlayNext**: A fade command. Set the parameter value to 1 to fade out the current main audio playlist track and fade in the designated audio file.
- **CrossfadeCurrentPlayNext**: A crossfade command. Set the parameter value to 1 to force an immediate crossfade between the current main audio playlist track and the designated audio file.

The following diagram illustrates how some of these timing parameters work together:



Example: The following example illustrates a simple crossfade between audio tracks.

```
a = CreateObject("roAudioPlayerMx")

track1 = CreateObject("roAssociativeArray")
track1["Filename"] = "file1.mp3"
track1["FadeInLength"] = 4000
track1["FadeOutLength"] = 4000
track1["QueueNext"] = 1

track2 = CreateObject("roAssociativeArray")
track2["Filename"] = "file2.mp3"
track2["FadeInLength"] = 4000
track2["FadeOutLength"] = 4000
track2["QueueNext"] = 1

a.PlayFile(track1)
a.PlayFile(track2)
```

Interfaces: *ifMediaTransport*, *ifSetMessagePort*, *ifAudioControl*, *ifSetMessagePort*, *ifAudioControlMx*

The *ifMediaTransport* interface provides the following:

- `PlayFile(a As Object) As Boolean`
- `Stop() As Boolean`
- `Play() As Boolean`
- `Pause() As Boolean`
- `Resume() As Boolean`

- `SetLoopMode(a As Boolean) As Boolean`
- `GetPlaybackStatus() As Object`

The *ifSetMessagePort* interface provides the following:

- `SetPort(a As Object)`

The *ifAudioControl* interface provides the following:

- `MapStereoOutput(a As Integer) As Boolean`
- `SetVolume(a As Integer) As Boolean`
- `SetChannelVolumes(a As Integer, b As Integer) As Boolean`
- `SetAudioOutput(a As Integer) As Boolean`
- `SetAudioMode(a As Integer) As Boolean`
- `SetAudioStream(a As Integer) As Boolean`
- `SetUsbAudioPort(a As Integer) As Boolean`
- `SetSpdifMute(a As Boolean) As Boolean`
- `MapDigitalOutput(a As Integer) As Boolean`
- `StoreEncryptionKey(a As String, b As String) As Boolean`
- `StoreObfuscatedEncryptionKey(a As String, b As String) As Boolean`
- `SetStereoMappingSpan(a As Integer) As Boolean`
- `ConfigureAudioResources() As Boolean`
- `SetPcmAudioOutputs(a As Object) As Boolean`
- `SetCompressedAudioOutputs(a As Object) As Boolean`

The *ifIdentity* interface provides the following:

- `GetIdentity() As Integer`

The *ifAudioControlMx* interface provides the following:

- `SetDecoderCount(a As Integer) As Boolean`

roCanvasWidget

This object composites background color, text, and images into a single rectangle, allowing you to layer images on a z-axis. Like any other widget, *roCanvasWidget* is created with an *roRectangle* to set its size and position on the screen.

Interfaces: *ifCanvasWidget*

The *ifCanvasWidget* interface provides the following:

- `Hide()`: Hides the widget.
- `Show()`: Shows the widget.
- `SetLayer(object content, int z-level)`: Sets the contents of a layer within the widget. The lowest z-level is drawn first, and the highest z-level is drawn last. The object content is described below.
- `ClearLayer(int z-level)`: Clears the specified layer.
- `Clear()`: Clears all of the layers.
- `EnableAutoRedraw(bool enable)`: Enables or disables the automatic redrawing of the widget.
  - When this function is enabled, each call to `SetLayer`, `ClearLayer`, or `Clear` results in a redraw. If you need to change multiple layers, then you should disable auto redraw while calling the `SetLayer` function.
  - `SetLayer` enables or disables redrawing of the widget when layer content is changed. When auto-redraw is enabled, each call to `SetLayer`, `ClearLayer`, or `Clear` results in a redraw. To batch multiple updates together, you should first suspend drawing using `EnableAutoRedraw(false)`, then make the changes to the content, and finally re-enable drawing using `EnableAutoRedraw(true)`. The redraw happens in a separate thread, so `EnableAutoRedraw` returns almost immediately.

Object Content

The content specified in each layer can consist of one or more objects. Each object is defined by an *roAssociativeArray*. If there is more than one object, then each is placed into an *roArray* prior to passing to the `SetLayer` function. Currently, there are four object types:

1. *Background color*
   - `color`: The `#[aa]rrggbb` hex value of the background color
   - `targetRect`: A target rectangle, which is another *roAssociativeArray* consisting of x, y, w, and h values. These values are relative to the top left corner of the widget.

2. *Text*
   - `text`: A string of text to display
   - `targetRect`: The rectangle in which the text is displayed
   - `textAttrs`: An *roAssociativeArray* containing attributes to be applied to the text. The attributes can be any of the following:
     - `font`: Small/medium/large/huge
     - `fontSize`: A point size that is used directly when creating the font. If the value is set to 0, then the font automatically resizes to fit the `targetRect`.
     - `fontfile`: The filename for a non-system font to use
     - `hAlign`: The left/center/right alignment of the text on a line
     - `vAlign`: The top/center/bottom alignment of the text perpendicular to the line
     - `rotation`: The 0/90/180/270 degree rotation of the text
     - `color`: The `#[aa]rrggbb` hex value of the text

3. *Image*

- `filename`: The filename of an image
- `targetRect`: The rectangle in which the image is displayed. The image will be automatically resized to fit into the target area.
- `sourceRect`: The source rectangle to clip from a source image
- `compositionMode`: Enter either `source` or `source_over.` The latter alpha blends with underlying objects. The former replaces the underlying values completely.

4. *QR Codes*

Note: *QR (quick response) codes appear as squares of black dots on a white background. They are used to encode URLs, email addresses, etc, and they can be scanned using readily available software for smart phones. Although the codes usually appear as black on white, you can, in theory, use any two contrasting colors.*

- `targetRect`: The rectangle in which the QR code is displayed
    - Regardless of the aspect ratio of this rectangle, the QR code itself will always be squared with the background color that fills the gaps.
- `QrCode` (simple form): Contains the string to encode into the QR code.
- `QrCode` (complex form): Contains an array of parameters for the QR code. The parameters can be any of the following:
    - `color`: The foreground color in the QR code (the default is black)
    - `backgroundColor`: The background color in the QR code (the default is white)
    - `rotation`: 0/90/180/270 degree rotation of the code. The code will scan regardless of rotation.
    - `qrText`: Contains the text to encode into the QR code.

Example: This code contains most of the *roCanvasWidget* features outlined above:

```
rect=CreateObject("roRectangle", 0, 0, 1920, 1080)
cw=CreateObject("roCanvasWidget", rect)

aa=CreateObject("roAssociativeArray")
aa["text"] = "Primal Scream"
aa["targetRect"] = { x: 280, y: 180, w: 500, h: 30 }
aa["textAttrs"] = { Color:"#AAAAAA", font:"Medium", HAlign:"Left", VAlign:"Top"}

aa1=CreateObject("roAssociativeArray")
aa1["text"] = "Movin' on up, followed by something else, followed by something else,
followed by something else, followed by something else"
aa1["targetRect"] = { x: 282, y: 215, w: 80, h: 500 }
aa1["textAttrs"] = { Color:"#ffffff", font:"Large", fontfile:"usb1:/GiddyupStd.otf",
HAlign:"Left", VAlign:"Top", rotation:"90"}

array=CreateObject("roArray", 10, false)
array.Push({ color: "5c5d5f" })
array.Push({ filename: "transparent-balls.png" })
array.Push(aa)

aa2=CreateObject("roAssociativeArray")
aa2["filename"] = "transparent-balls.png"
aa2["CompositionMode"] = "source_over"
aa2["targetRect"] = { x: 400, y: 200, w: 200, h: 200 }

aa3=CreateObject("roAssociativeArray")
```

```
aa3["QrCode"] = "www.brightsign.biz"
aa3["targetRect"] = { x: 100,  y: 100,  w: 400, h: 400  }

aa4=CreateObject("roAssociativeArray")
aa4["QrCode"] = { qrText:"www.brightsign.biz", rotation:"90" }
aa4["targetRect"] = { x: 1200,  y: 100,  w: 400, h: 600  }

aa5=CreateObject("roAssociativeArray")
aa5["QrCode"] = { color:"#964969", backgroundColor:"#FFFF77",
qrText:"www.brightsign.biz", rotation:"180" }
aa5["targetRect"] = { x: 100,  y: 600,  w: 400, h: 400  }

cw.Show()
cw.EnableAutoRedraw(0)
cw.SetLayer(array, 0)
cw.SetLayer(aa1, 1)
cw.SetLayer(aa1, 2)
cw.SetLayer(aa3, 3)
cw.SetLayer(aa4, 4)
cw.SetLayer(aa5, 5)
cw.EnableAutoRedraw(1)

cw.ClearLayer(0)
```

roClockWidget

This object places a clock on the screen. It has no extra interface, only construction arguments.

Interfaces: *ifTextWidget*, *ifWidget*

Object creation:

```
CreateObject("roClockWidget", rect As roRectangle, res As roResourceManager, display_type
As Integer)
```

- `Rect`: The rectangle in which the clock is displayed. The widget picks a font based on the size of the rectangle.
- `display_type`: Use 0 for date only, and 1 for clock only. To show both on the screen, you need to create two widgets.

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 into the widget, which uses the following resources within “resources.txt” to display the time and 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|November|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)
```

See the entry for *roTextWidget* for a description of *ifTextWidget*.

The *ifWidget* interface provides the following:

- `GetFailureReason() As String`
- `SetForegroundColor(a As Integer) As Boolean`
- `SetBackgroundColor(a As Integer) As Boolean`
- `SetFont(a As String) As Boolean`
- `SetBackgroundBitmap(a As String, b As Boolean) As Boolean`
- `SetSafeTextRegion(a As Object) As Boolean`
- `Hide() As Boolean`
- `Show() As Boolean`

roHtmlWidget

This object embeds the WebKit HTML renderer. You can use multiple instances of *roHtmlWidget* at the same time.

Object creation: Like other widgets, an *roHtmlWidget* is created with an *roRectangle*, which specifies the size and position that the widget occupies on the screen.

Interfaces: *ifHtmlWidget*, *ifMessagePort*

The *ifHtmlWidget* interface provides the following:

- `GetFailureReason() As String`: : Gives more information when a member function returns False.
- `Hide() As Boolean`: Hides the widget.
- `Show() As Boolean`: Shows the widget.
- `SetURL(a As String) As Boolean:` Displays content from the specified URL.

Note
When using `SetUrl` to retrieve content from local storage, you do not need to specify the full file path: `SetUrl("file:/example.html")`. If the content is located somewhere other than the current storage device, you can specify it within the string itself. For example, you can use the following syntax to retrieve content from a storage device inserted into the USB port when the current device is an SD card: `SetUrl("file:///USB1:/example.html")`.

- `MapFilesFromAssetPool(a As Object, b As Object, c As String, d As String) As Boolean`: Sets the mapping between the URL space and the pool files.
- `SetZoomLevel(a as Float)`: Adjusts the scale factor for the displayed page (the default equals 1.0).
- `EnableSecurity() As Boolean`: Enables security checks by the Webkit. Setting this method to False disables security checks.

- `EnableMouseEvents() As Boolean`: Enables response to button presses if True. Setting this method to False (the default) disables this feature.
- `SetPortriat() As Boolean`: Sets the screen orientation to portrait if True. If this method is False (the default), the screen is oriented as a landscape.
- `SetAlpha (a As Integer)`: Sets the overall alpha level for the widget (the default equals 255).
- `Scroll (x As Integer, y As Integer)`: Scrolls the viewport to a new location.
- `EnableScrollbars() As Boolean`: Enables automatic scrollbars for content that does not fit into the viewport if True. Setting this method to False (the default) disables this feature.
- `AddFont (filename As String)`: Makes a font available for text rendering. The `AddFont()` method can be used to supply additional or custom typefaces for the WebKit renderer. These should be supplied in the form of TTF files, and the filename should be passed as the argument to `AddFont().`
- `SetAppCacheDir(file_path As String)`: Sets the directory to use for storing the application cache (which services <`html manifest=”example.appcache”`> tags). The file path is passed to the method as a string (e.g. “SD:/appcache”).
- `SetAppCacheSize(maximum As Integer)`: Sets the maximum size (in bytes) for the application cache. Changing the storage size of the application cache will clear the cache and rebuild the cache storage. Depending on database-specific attributes, you will only be able to set the size in units that are equal to the page size of the database, which is established at creation. These storage units will occur only in the following increments: 512, 1024, 2048, 4096, 8192, 16384, 32768.
- `EnableJavascript(a As Boolean) As Boolean`
- `SetUserAgent(a As String) As Boolean`

The *ifMessagePort* interface provides the following:

- `SetPort (a As Object)`

An *roMessagePort* can be attached to an *roHtmlWidget*. It will then receive *roHtmlWidgetEvent* objects when something happens to the parent widget. An *roAssociativeArray* functions as the payload of the *roHtmlWidgetEvent*, and the payload can be retrieved using the `GetData()` method. Within the associative array, the `reason` key identifies the cause of the event. The `reason` key can return the following values:

- `load-started`: The WebKit has started loading a page.
- `load-finished`: The WebKit has completed loading a page.
- `load-error`: The WebKit has failed to load a page. The `uri` key identifies the failing resource, and the `message` key provides some explanatory text.

Filename Mapping

HTML content that has been deployed via BrightAuthor will typically reside in the pool and have encrypted SHA1-based filenames. A mapping mechanism is required to allow any relative URIs contained in the HTML content to continue working and to locate the appropriate resources in their respective pool locations.

An `roHtmlWidget.MapFilesFromAssetPool()` method can be used to bind part of the resource URI space onto pool locations, as long as it is used with the following: an roAssetPool object containing some assets, an *roAssetCollection* object identifying them, and two semi-arbitrary strings (URI_PREFIX and POOL_PREFIX).

Any URI in the form `“file:/[URI_PREFIX][RESOURCE_ID]”` will be rewritten into the form `“[POOL_PREFIX][RESOURCE_ID]”`. It will then be located in the pool as if that name had been passed to the *roAssetPoolFiles.GetPoolFilePath()* method. This binding occurs for every instance of *roHtmlWidget*, so different mappings can be used for different bundles of content.

roImageBuffer

Interfaces: *ifImageBufferControl*

The *ifImageBufferControl* interface provides the following:

- `DisplayBuffer(a As Integer, b As Integer) As Boolean`

roImagePlayer

This object displays static bitmap images on the video display.

Interfaces: *ifImageControl*

The *ifImageControl* interface provides the following:

- `DisplayFile(image_filename As String) As Boolean`
- `DisplayFile(parameter As roAssociativeArray) As Boolean`
- `PreloadFile(filename As String) As Boolean`
- `PreloadFile(parameter As roAssociativeArray) As Boolean`
- `DisplayPreload As Boolean`
- `StopDisplay As Boolean`: Removes an image from the display.
- `DisplayFileEx(filename As String, mode As Integer, x As Integer, y As Integer) As Boolean`
- `PreloadFileEx(filename As String, mode As Integer, x As Integer, y As Integer) As Boolean`
- `SetDefaultMode(mode As Integer) As Boolean`
- `SetDefaultTransition(transition As Integer) As Boolean`
- `SetRectangle(r As roRectangle) As Void`
- `OverlayImage(a As String, b As Integer, c As Integer) As Boolean`
- `GetRectangle() As Object`
- `CreateTestHole(a As Object) As Boolean`
- `SetTransitionDuration(a As Integer) As Boolean`
- `DisplayBuffer(a As Object, b As Integer, c As Integer) As Boolean`

The simplest way to use *roImagePlayer* is to make calls to “DisplayFile” with the filename as a String. Alternatively, you can use *PreloadFile/DisplayPreload* to have more control.

PreloadFile loads the file into an off-screen memory 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; and 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* calls 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 both x and y can be set to -1, which uses the center of the image as the point to position nearest to the center.

SetDefaultMode sets the mode used for *DisplayFile* and *PreloadFile*. If *SetDefaultMode* is not called, then the default mode is set to 0 (equivalent to centered without scaling) when the object is created.

Currently, *image_filename* must point to a PNG, JPEG, or a 8-bit, 24-bit, or 32-bit BMP file.

The supported Display Modes are listed here:

- 0 - Center image: No scaling takes place. Only cropping occurs 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 completely fills the screen, 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. The following are available transitions:

- 0 - No transition, immediate blit.

- 1 to 4 - 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.
- 20 to 23 – Slides entire screen from top, bottom, left, and right.
- 24, 25 – Scales old image in, then the new one out again (this works as a pseudo “rotation” around a vertical and horizontal axis, respectively).
- 26 to 29 – New image expands onto screen from right, left, bottom, and top, respectively.

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 use 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
```

The following example script uses preloaded images to improve the UI speed when the user hits a key on the keyboard. As soon as a key is struck, the display switches to the new image, which has already been preloaded. The only possible delay occurs if the key is hit while the image is preloading. In this case, the image will display as soon as it is loaded.

```
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
```

roImageWidget

This object can be used in place of *roImagePlayer* in cases where the image is displayed within a rectangle. Using a *roImageWidget* can result in more pleasing aesthetics for image player creation. Beyond this, *roImageWidget* behaves identically to *roImagePlayer*.

When displaying *roImagePlayer* within a rectangle, the following code is used:

```
rectangle = CreateObject("roRectangle", 0, 0, 1024, 768)
i = CreateObject("roImagePlayer")
i.SetRectangle(rectangle)
```

When utilizing an *roImageWidget*, the following code is used:

```
rectangle = CreateObject("roRectangle", 0, 0, 1024, 768)
i = CreateObject("roImageWidget", rectangle)
```

Interfaces: *ifImageControl*

See *roImagePlayer* for a description of *ifImageControl* and its attendant methods.

We have added some overloaded *PreloadFile* and *DisplayFile* functions. These take an *roAssociativeArray* as a parameter. The *roAssociativeArray* stores various options to be passed. They must be used when displaying images across multiple screens in an array, or displaying a portion of an image though they can also be used in place of the original function calls.

Example: This code uses *PreloadFile* for a multiscreen display:

```
i=CreateObject("roImagePlayer")
a=CreateObject("roAssociativeArray")
a["Filename"] = "test.jpg"
a["Mode"] = 1
a["Transition"] = 14
a["MultiscreenWidth"] = 3
a["MultiscreenHeight"] = 2
a["MultiscreenX"] = 0
a["MultiscreenY"] = 0
i.PreloadFile(a)
i.DisplayPreload
```

The *filename*, *mode*, and *transition* are the same values as documented above, but the multiscreen parameters are new. The *MultiscreenWidth* and *MultiscreenHeight* parameters specify the width and height of the multiple screen matrix. For example, 3x2 would be three screens wide and two screens high. The *MultiscreenX* and *MultiscreenY* specify the position of the current screen within that matrix.

In the above case, on average only 1/6 of the image is drawn on each screen, though the image mode still applies so that depending on the shape of the image, it may have black bars on the side screens. It is relatively simple, therefore, for an image widget to display part of an image based on its position in the multiscreen array. The following are default values for the parameters:

```
Mode = 0
Transition = 0
MultiscreenWidth = 1
MultiscreenHeight = 1
MultiscreenX = 0
```

```
MultiscreenY = 0
```

Example: This code uses *DisplayFile* for displaying a portion of an image:

```
i=CreateObject("roImagePlayer")
a=CreateObject("roAssociativeArray")
a["Filename"] = "test.JPG"
a["Mode"] = 0
a["SourceX"] = 600
a["SourceY"] = 600
a["SourceWidth"] = 400
a["SourceHeight"] = 400
i.DisplayFile(a)
```

This displays just a portion of the image test JPG starting at coordinates *SourceX, SourceY*, and *SourceWidth* by *SourceHeight* in size. The *viewmode* is still honored as if it were displaying the whole file.

roRectangle

This object is created with several parameters:

```
CreateObject("roRectangle", x As Integer, y As Integer, width As Integer, height As
Integer)
```

Interfaces: *ifRectangle*

The interface *ifRectangle* provides the following:

- `SetX(x As Integer) As Void`
- `SetY(y As Integer) As Void`
- `SetWidth(width As Integer) As Void`
- `SetHeight(height As Integer) As Void`
- `GetX As Integer`
- `GetY As Integer`
- `GetWidth As Integer`
- `GetHeight As Integer`

SetRectangle calls honor the view mode/aspect ratio conversion mode set up by the user. If the user has set the videoplayer for letterboxing, it will occur if the video does not fit exactly into the new rectangle.

roShoutcastStream

Object creation: *roShoutcastStream* takes a URL object, a maximum buffer size (in seconds), and an initial buffering duration (in seconds).

```
CreateObject("roShoutcastStream", url_transfer, buffer size, buffer duration)
```

Interfaces: *ifShoutcastStream*, *ifSetMessagePort*, *ifSourceIdentity*

The *ifShoutcastStream* interface provides the following:

- `GetUrl() As String`
- `GetBufferedDuration() As Integer`
- `GetTimeSinceLastData() As Integer`
- `GetCurrentMetadata() As String`
- `Rebuffer() As Boolean`
- `AsyncSaveBuffer(a As String) As Boolean`
- `RestartBufferRecord() As Boolean`

The *ifSetMessagePort* interface provides the following:

- `SetPort(a As Object)`

The *ifSourceIdentity* interface provides the following:

- `GetSourceIdentity() As Integer`
- `SetSourceIdentity(a As Integer)`

roShoutcastStreamEvent

Interfaces: *ifInt*, *ifSourceIdentity*

The *ifInt* interface provides the following:

- `GetInt() As Integer`
- `SetInt(a As Integer)`

The *ifSourceIdentity* interface provides the following:

- `GetSourceIdentity() As Integer`
- `SetSourceIdentity(a As Integer)`

roTextField

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 object is created with several parameters:

```
CreateObject("roTextField", xpos As Integer, ypos As Integer, width_in_chars As Integer,
height_in_chars As Integer, metadata As Object)
```

- `xpos`: The horizontal coordinate for the top left of the text field.
- `ypos`: The vertical coordinate for the top left of the text field. The top of the screen is equivalent to zero.
- `width_in_chars`: The width of the text field in character cells.
- `height_in_chars`: The height of the text field in character cells.
- `metadata`: An optional *roAssociativeArray* containing extra parameters for the text field. You can pass zero if you do not require this.

Note: *In TV modes a border around the screen may not be displayed due to overscanning. You may want to use the* roVideoMode *object functions* GetSafeX *and* GetSafeY *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) for red, green and blue in the form `&Hrrggbb`.
- "TextColor": The color of the text as an integer specifying eight bits (for each) for 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).

Interfaces: *ifTextField*, *ifStreamSend*

The *ifTextField* interface provides the following:

- `Cls As Void`: Clears the text field.
- `GetWidth As Integer`: Returns the width of the text field
- `GetHeight As Integer`: Returns the height of the text field.
- `SetCursorPos(x As Integer, As Integer) As Void`: Moves the cursor to the specified position. Subsequent output will appear at this position.
- `GetValue As Integer`: Returns the value of the character currently under the cursor.

The *ifStreamSend* interface provides the following:

- `SendByte(byte As Integer) As Void`: Writes the character indicated by the specified number at the current cursor position within the text field. It then advances the cursor.
- `SendLine(string As String) As Void`: Writes the characters specified at the current cursor position followed by the end-of-line sequence.
- `SendBlock(string As String) As Void`: Writes the characters specified at the current cursor position and advances the cursor to one position beyond the last character.
- `SetSendEol(string As String) As Void`: Sets the sequence sent at the end of a *SendLine* request. You should leave this at the default value of “chr(13)” for normal use.

Note: *The ifStreamSend interface is also described in the section documenting the various file objects. The interface is described again here in a manner more specific to the roTextField object.*

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 for more details.

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; and positions 16-23 are in the last row).

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

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"
```

roTextWidget

This object is used to place text on the screen.

Object creation:

`CreateObject("roTextWidget", r As roRectangle, line_count As Integer, text_mode As Integer, pause_time As Integer)`

- `r` : Create an *roRectangle* that contains the text.
- `line_count`: Determine the number of lines of text to show in the rectangle.
- `text_mode`: Use 0 for an animated view similar to teletype, 1 for static text, and 2 for simple text with no queue of strings.
- `pause_time`: Determine how long each string is displayed prior to displaying the next string.

`CreateObject("roTextWidget", r As roRectangle, line_count As Integer, text_mode As Integer, array As roAssociativeArray)`

- `r` : Create an *roRectangle* that contains the text.
- `line_count`: Determine the number of lines of text to show in the rectangle.
- `text_mode`: Use 0 for an animated view similar to teletype, 1 for static text, and 2 for simple text with no queue of strings.
- `array`: Create an associative array that can include the following values:
    - "LineCount": Determine the number of lines of text to show in the rectangle.
    - "TextMode": Use 0 for an animated view similar to teletype, or use 1 for static text.
    - "PauseTime": Determine how long each string is displayed prior to displaying the next string. This does not apply to mode 2 *roTextWidget* objects, in which the strings on screen are updated immediately.
    - "Rotation": Determine the rotation of the text in the widget: 0 degrees (0), 90 degrees (1), 180 degrees (2), or 270 degrees (3).

    - “Alignment”: Determine the alignment of the text: left (0), center (1), or right (2).

Interfaces: *ifTextWidget*, *ifWidget*

The *ifTextWidget* interface provides the following:

- `PushString(str As String) As Boolean`: Adds the string to the list of strings to display in modes 0 and 1. Strings are displayed in order, and when the end is reached, the object loops, returning to the beginning of the list. In mode 2, the string is displayed immediately.
- `PopStrings(number_of_string_to_pop As Integer) As Boolean`: Pops strings off the front of the list (using “last in, first out” ordering) in modes 0 and 1. This occurs the next time the widget wraps so that strings can be added to and removed from the widget seamlessly. In mode 2, the string is cleared from the widget immediately.
- `GetStringCount As Integer`: Returns the number of strings that will exist once any pending pops have taken place.
- `Clear As Boolean`: Clears the list of strings, leaving the widget blank and able to accept more *PushString* calls.

This *ifWidget* interface provides the following:

- `SetForegroundColor(color As Integer) As Boolean`
- `SetBackgroundColor(color As Integer) As Boolean`: Sets the background *color* in ARGB format.
- `SetFont(font_filename As String) As Boolean`: Sets the *font_filename* using a TrueType font (for example, SD:/Ariel.ttf).
- `SetBackgroundBitmap(background_bitmap_filename As String, stretch As Boolean) As Boolean`: Sets the background bitmap. If *stretch* is True, then the image is stretched to the size of the window.
- `SetSafeTextRegion(rect As roRectangle) As Boolean`: Specifies the rectangle within the widget where the text can be drawn safely.
- `Show As Boolean`: Displays the widget. After creation, the widget is hidden until *Show* is called.
- `Hide As Boolean`: Hides the widget.
- `GetFailureReason() As String`: Yields additional useful information if a function return indicates an error.

The top 8 bits of the color value are “alpha,” which has no effect on the foreground text color, but does affect the widget background color. Zero is equivalent to fully transparent and 255 to fully non-transparent. This feature allows effects similar to subtitles. For example, you can create a semi-transparent black box containing text over video.

Modes 0 and 1 are useful for displaying RSS feeds and ticker-type text. However, for dynamic data where immediate screen updates are required, mode 2 is more appropriate. Mode 2 allows text to be drawn immediately to the screen.

roVideoEvent, 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:

1 `Undefined`: Player is in an undefined state.
2 `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 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`: The media item has completed playback.
9 `Transitioning`: Player is preparing new media item.
10 `Ready`: Player is ready to begin playing.
11 `Reconnecting`: Player is reconnecting to the stream.
12 `TimeHit`: A particular timecode is hit. See *roVideoPlayer*.

Interfaces: *ifInt*, *ifData*

The *ifInt* interface contains the event ID enumerated above and provides the following:

- `GetInt As Integer`
- `SetInt(a As Integer)`
- `GetSourceIdentity() As Integer`
- `SetSourceIdentity() As Integer`

The ifData interface contains userdata and provides the following:

- `GetData As Integer`

- `SetData(a As Integer)`

Example:

```
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
```

roVideoInput

This object allows playback of video supplied by a video capture dongle.

roVideoInput is created with no parameters:

```
CreateObject("roVideoInput ")
```

Interfaces: *ifVideoInput*

The *ifVideoInput* interface provides the following:

- `GetStandards As roArray(String)`
- `GetInputs As roArray(String)`: These return an array of strings describing the various inputs and video standards that the video capture device supports. The following are the possible standards that can be returned: PAL-D/K, PAL-G, PAL-H, PAL-I, PAL-D, PAL-D1, PAL-K, PAL-M, PAL-N, PAL-Nc, PAL-60, SECAM-B/G, ECAM-B, SECAM-D, SECAM-G, SECAM-H, SECAM-K, SECAM-K1, SECAM-L, SECAM-LC, SECAM-D/K, NTSC-M, NTSC-Mj, NTSC-443, NTSC-Mk, PAL-B and PAL-B1. Inputs returned are s-video and composite.
- `SetStandard(As String) As Boolean`
- `GetCurrentStandard As String`
- `SetInput(As String) As Boolean`
- `GetCurrentInput As String`: Use the above to get and set the input and video standard.
- `GetControls As roArray(String)`: Returns the possible controls on the input. These include “Brightness,” “Contrast,” “Saturation,” “Hue,” and others.
- `SetControlValue(control_name As String, value As Integer) As Boolean`: Sets the value of the specified control.

- `GetCurrentControlValue(control_name As String) As roAssociativeArray`: Returns an associative array with 3 members: “Value,” “Minimum,” and “Maximum.” “Value” is the current value, and the possible range is specified by “Minimum” and “Maximum.”
- `GetFormats() As Object`
- `SetFormat(a As String, b As Integer, c As Integer) As Boolean`
- `GetCurrentFormat() As String`

Example: This script creates a full-screen display with the video capture dongle as the video source.

```
v=CreateObject("roVideoPlayer")
i=CreateObject("roVideoInput")
p=CreateObject("roMessagePort")

vm=CreateObject("roVideoMode")
vm.SetMode("1280x720x60p")

r = CreateObject("roRectangle", 0, 0, 1280, 720)
v.SetRectangle(r)

i.SetInput("s-video")
i.SetStandard("ntsc-m")

v.PlayEx(i)
```

roVideoMode

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

Interfaces: *ifVideomode*, *ifSetMessagePort*

The *ifVideoMode* interface provides the following:

- `SetMode(mode As String) As Boolean`
- `GetResX As Integer`: Gets the width of the display for the current video mode.
- `GetResY As Integer`: Gets the height of the display for the current video mode.
- `GetSafeX As Integer`: Gets the left coordinates for the start of the "safe area." For modes that are generally displayed with no overscan, this will be zero.
- `GetSafeY As Integer`: Gets the top coordinates for the start of the "safe area." For modes that are generally displayed with no overscan, this will be zero.
- `GetSafeWidth As Integer`: Gets the width of the "safe area." For modes that are generally displayed with no overscan, this will return the same as *GetResX*.
- `GetSafeHeight As Integer`: Gets the height of the "safe area." For modes that are generally displayed with no overscan, this will return the same as *GetResY*.

Note: *More information about safe areas can be found here:*

- http://en.wikipedia.org/wiki/Safe_area
- http://en.wikipedia.org/wiki/Overscan_amounts

- `SetBackgroundColor(a As Integer) As Boolean`: Specifies the background color using an `#rrggbb` hex value.
- `SetPowerSaveMode(power_save_enable As Boolean) As Boolean`: Turns off the syncs for VGA output and the DAC output for component video. This will cause some monitors 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, which you find on the support page, for more information.

If the video mode specified in *SetMode* is different from the object's current video mode, the unit will reboot and set its video mode to the new setting during system initialization.

- `SetMultiscreenBezel(x_pct As Integer, y_pct As Integer) As Boolean:` Adjusts the size of the bezel used in calculations when using multiscreen displays for video and images. It allows users to compensate for the width of their screen bezels in multiscreen configurations. The calculations for the percentages are as follows:

  x_percentage = (width_of_bezel_between_active_screens / width_of_active_screen) * 100

  y_percentage = (height_of_bezel_between_active_screens / height_of_active_screen) * 100

  The bezel measurement is therefore the total of the top and bottom bezels in the y case, or the left and right bezels in the x case. When this value is set correctly, images spread across multiple screens take account of the bezel widths, leading to better alignment of images.

- `GetEdidIdentity(a As Boolean) As Object:`Returns an associative array with EDID information from a compatible monitor/television connected over HDMI. These are the possible parameters:
  - `serial_number_string`
  - `year_of_manufacture`
  - `monitor_name`
  - `manufacturer`
  - `text_string`
  - `serial_number`

  - `product`
  - `week_of_manufacture`

  The system will generate an *roHdmiEdidChanged* event when an HDMI cable is hotplugged and the EDID information changes. Calling `GetEdidIdentity(1)` at this point retrieves the new EDID information.

The *ifSetMessagePort* interface provides the following:

- `SetPort(obj As Object) As Void`

The following are supported modes that can be passed to *SetMode* on the HD110, HD210, HD210w, HD410, HD810, HD1010, HD1010w, HD120, HD220 and HD1020:

- “auto”
- “640x480x60p”
- “800x600x60p” (The HD210w and HD1010w support “800x600x75p” instead)
- “1024x768x60p” (The HD210w and HD1010w support “1024x768x75p” instead)
- “1280x768x60p”
- “1280x800x60p
- “1360x768x60p”
- “720x576x50p”
- “720x480x60p”
- “1280x720x50p”
- “1280x720x59.94p”
- “1280x720x60p”
- “1920x1080x50i”
- “1920x1080x59.94i”
- “1920x1080x60i”
- “1920x1080x29.97p”

- “1920x1080x50p”
- “1920x1080x60p”

If the mode is set to “auto,” the BrightSign will try to determine the best video mode to use based on connected hardware. The algorithm is as follows:

1. Try VGA first – If VGA is attached, use the best mode as reported by the monitor that BrightSign supports.
2. Try HDMI next – If HDMI is attached, use the best mode as reported by the monitor that BrightSign supports.
3. Default to 1024x768x75p.

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 a different thread from the script. The script will continue to run while video is playing. Decoded video will be scaled to the output resolution specified by *roVideoMode*.

Interfaces: *ifIdentity*, *ifSetMessagePort*, *ifAudioControl*, *ifAudioAuxControl*, *ifVideoControl*, *ifMediaTransport*.

The *ifIdentity* interface provides the following:

- `GetIdentity() As Integer`

The *ifSetMessagePort* interface provides the following:

- `SetPort(obj As Object)As Void`
- `SetPort(a As Object)`

See *roAudioPlayer* for an explanation of *ifAudioControl*.

The *ifAudioAuxControl* interface provides the following:

- `MapStereoOutputAux(a As Integer) As Boolean`
- `SetVolumeAux(a As Integer) As Boolean`
- `SetChannelVolumesAux(a As Integer, b As Integer) As Boolean`
- `SetAudioOutputAux(a As Integer) As Boolean`
- `SetAudioModeAux(a As Integer) As Boolean`
- `SetAudioStreamAux(a As Integer) As Boolean`
- `SetUsbAudioPortAux(a As Integer) As Boolean`

The *ifVideoControl* interface provides the following:

- `PlayStaticImage(filename As String) As Boolean`
- `SetViewMode(mode As Integer) As Boolean`
- `SetRectangle(r As roRectangle) As Void`
- `EnableSafeRegionTrimming(a As Boolean) As Boolean`

The *ifMediaTransport* interface provides the following:

- `PlayFile(filename As String) As Boolean`
- `PlayFile(parameters As AssociativeArray) As Boolean`
- `PreloadFile(parameters As AssociativeArray) As Boolean`
- `Stop As Boolean`
- `Play As Boolean`
- `SetLoopMode(mode As Integer) As Boolean`
- `ClearEvents As Boolean`
- `AddEvent(userdata As Integer, time_in_ms As Integer) As Boolean`
- `StopClear As Boolean`
- `Pause() As Boolean`
- `Resume() As Boolean`
- `PlayEx(a As Object) As Boolean`

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

view_mode values:

0 – Scale to fill (default). The aspect ratio can change.

1 – Letterboxed and centered. The aspect ratio is maintained, and the video has black borders.

2 – Full 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 the 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. This will fail only when a widescreen monitor displays a 4:3 output resolution such as 800x600 across the whole screen (i.e. the monitor does not 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 that 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*.

Example: This script uses timecode events. The script prints out 2, 5, and 10 into the video at 2 seconds, 5 seconds, and 10 seconds. 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("SD:/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
```

Calling *PlayStaticImage* displays an image on the video layer. The image is stretched to fill the video rectangle.

Multiscreen video playback

We have also added some overloaded *PreloadFile* and *PlayFile* functions. These take a *roAssociativeArray* as a parameter, which stores all the various options to be passed in. They must be used when displaying images across multiple screens in an array, or displaying windowed portions of a video, though they can also be used in place of the original function calls.

Example: This script uses the *PreloadFile* for a multiscreen display:

```
v=CreateObject("roVideoPlayer")
a=CreateObject("roAssociativeArray")
a["Filename"] = "test.ts"
a["MultiscreenWidth"] = 3
a["MultiscreenHeight"] = 2
a["MultiscreenX"] = 0
a["MultiscreenY"] = 0
v.PreloadFile(a)
…
…
v.Play()
```

The filename is the same as documented earlier, but the multiscreen parameters are new. *MultiscreenWidth* and *MultiscreenHeight* specify the width and height of the multiple-screen matrix. For example, 3x2 would be 3 screens wide and 2 high. *MultiscreenX* and *MultiscreenY* specify the position of the current screen within that matrix. In the case above, on average only 1/6th of the video is drawn on each screen (though the view mode still applies), so depending on the shape of the video, it may have black bars on the side screens. In this way, it is relatively simple for a video player to display part of an image based on its position in the multiscreen array.

PreloadFile does all of the preliminary work to get ready to play the specified video clip, including stopping the playback of the previous video file. The call to “Play” starts the playback. This is good for synchronizing video across multiple players as they can all be prepared ready to play and then will immediately start playing when the “Play” command is issued. This reduces synchronization latencies.

The following are the default values for the parameters:
MultiscreenWidth = 1
MultiscreenHeight = 1
MultiscreenX = 0
MultiscreenY = 0

Example: Here is a script using *PlayFile* for displaying a portion of a video:

```
v=CreateObject("roVideoPlayer")
a=CreateObject("roAssociativeArray")
a["Filename"] = "test.ts"
a["SourceX"] = 100
a["SourceY"] = 100
a["SourceWidth"] = 1000
a["SourceHeight"] = 500
v.PlayFile(a)
```

This displays a windowed portion of the video test.ts starting at coordinates *SourceX*, *SourceY*, and *SourceWidth* by *SourceHeight* in size. The *viewmode* is still honored as if displaying the whole file.

RF Channel Scanning

The `PlayFile()` method can be used for channel scanning and handling functionality similar to *roChannelManager*. To use `PlayFile()` for channel scanning, pass an *roAssociativeArray* with the following possible parameters:

- `VirtualChannel`
- `RfChannel`
- `SpectralInversion`
    - `INVERSION_ON`
    - `INVERSION_OFF`
    - `INVERSION_AUTO`
- `ModulationType`
    - `QAM_64`
    - `QAM_256`
    - `QAM_AUTO`
    - `8VSB`
- `VideoCodec`
    - `MPEG1-Video`
    - `MPEG2-Video`
    - `MPEG4Part2-Video`
    - `H264`
    - `H264-SVC`
    - `H264-MVC`
    - `AVSC`
- `AudioCodec`
    - `MPEG-Audio`

    - AAC
    - AAC+
    - AC3
    - AC3+
    - DTS
- VideoPid
- AudioPid
- PcrPid

The VirtualChannel and RfChannel parameters must be present for *PlayFile()* to scan correctly. If you specify only these parameters, the player will scan the RF channel for a QAM/ATSC signal and attempt to retrieve the specified virtual channel from the results. The results from this action are cached so that subsequent calls to *PlayFile()* will take much less time. Providing the SpectralInversion and/or ModulationType parameters will further speed up the scanning process.

If all parameters are supplied, then no scanning is required and the player can tune to the channel immediately. If one or more of the optional parameters is missing, then the player must parse the transport stream metadata to find the appropriate values for the supplied VirtualChannel and RfChannel.

roTouchCalibrationEvent

Interfaces: *ifInt*, *ifIntOps*

The *ifInt* interface provides the following:

- `GetInt() As Integer`
- `SetInt(a As Integer)`

The *ifIntOps* interface provides the following:

- `ToStr() As String`

roTouchEvent

Interfaces: *ifInt*, *ifPoint*, *ifEvent*

The *ifInt* interface provides the following:

- `GetInt() As Integer`
- `SetInt(a As Integer)`

The *ifPoint* interface provides the following:

- `GetX() As Integer`
- `GetY() As Integer`
- `SetX(a As Integer)`
- `SetY(a As Integer)`

The *ifEvent* interface provides the following:

- `GetEvent() As Integer`
- `SetEvent(a As Integer)`

roTouchScreen

This object allows you to accept events from touch screen panels or mice. Not all touchscreens are supported. However, we are always working on more driver support. Please see this FAQ for a full list of supported touch screens, or contact sales@brightsign.biz if you want to see a specific touch-screen model supported.

roTouchScreen responds to the clicks of a USB mouse in the same way it responds to touch events on a touch screen.

To use a touch screen, follow these general steps:

1. Create an *roTouchScreen* object.
2. Use *SetPort* to communicate to 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 the message port.
4. Process the events.

Note: *If touch areas overlap such that a touch hits multiple regions, an event for each affected region will be sent.*

The *roTouchScreen* object 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. In all other cases, the *off* image is displayed. This allows buttons to be highlighted as the mouse cursor moves over them.

Interfaces: *ifTouchScreen*, *ifSetMessagePort*, *ifTouchScreenCalibration*, *ifSerialControl*

The *ifTouchScreen* interface provides the following:

- `SetResolution(x As Integer, y As Integer) As Void`
- `AddRectangleRegion(x As Integer, y As Integer, w As Integer, h As Integer, userid As Integer) As Void`
- `AddCircleRegion(x As Integer, y As Integer, radius As Integer, userid As Integer) As Void`
- `ClearRegions()`: 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.
- `GetDeviceName As String`
- `SetCursorPosition(x As Integer, y As Integer) As Void`
- `SetCursorBitmap(As String, x As Integer, y As Integer) As Void`
- `EnableCursor(on-off As Boolean) As Void`
- `EnableRollover(region_id As Integer, on_image As String, off_image As String, cache_image As Boolean, image_player As Object) As Void`: Enables a rollover for a touch region. The function accepts the ID of the touch region, as well as two strings specifying the names of the *on* and *off* bitmap images, a cache setting, and the image player that draws the rollover. The *cache_image* parameter simply tells the script whether to keep the bitmaps loaded in memory or not. This setting uses up memory very quickly, so we recommend that *cache_image* normally be set to 0.
- `EnableRegion(region_id As Integer, enabled As Boolean) As Void`: Enables or disables a rollover region. The function accepts the ID of the touch region, as well as 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 enable the current ones when required.
- `SetRollOverOrigin(region_id As Integer, x As Integer, y As Integer) As Void: Changes` 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 that are larger than the region can be drawn. The default requirement is that rollover bitmaps be the same size and position as the touch region (though, for circular regions, the bitmap is square). Note that the default origin for circular regions is **x - r, y - r** where **x, y** is the center and **r** is the radius.

- `IsMousePresent() As Boolean`: Returns whether a relative pointing device is attached or not.

Note: This does not work for absolute devices like touch screens.

- `EnableSerialTouchscreen(a As Integer) As Boolean`
- `SetSerialTouchscreenConfiguration(a As String) As Boolean`

The *ifSetMessagePort* interface provides the following:

- `SetPort(a As Object)`

The *ifTouchScreenCalibration* interface provides the following:

- `StartCalibration() As Boolean`
- `GetCalibrationStatus() As Integer`

The *ifSerialControl* interface provides the following:

- `SetBaudRate(a As Integer) As Boolean`
- `NotUsed1(a As String)`
- `SetMode(a As String) As Boolean`
- `NotUsed2(a As Boolean) As Boolean`

The *roTouchScreen* interface sends events of type *roTouchEvent*, which provides the following:

- *ifInt*: The *userid* of the touched region.
- *ifPoint*: The **x,y** coordinates of the touch point. This interface is not normally needed. *ifPoint* has two member functions: `GetX As Integer` and `GetY As Integer`.
- *ifEvent*: The mouse events. *ifEvent* has one member function: `GetEvent As Integer`.

Information about the cursor

The mouse cursor is a 32x32 pixel square, where each pixel can be one of 16 different colors. These colors are 16 bits, with 14 bits of color and 2 bits of alpha. If you use all of the alpha levels on all shades, then you limit the number of available shades to five (five shades at three alpha levels plus one fully transparent color gives 16). The colors are actually specified internally in YUV (6-4-4 bits respectively), but BrightScript supports setting the cursor from PNG, JPG, GIF or BMP.

Rules:

1. The 32x32 pixel square in which the cursor is drawn cannot be moved outside of the screen boundaries.
2. The 32x32 cursor can be moved around within the screen boundaries, but can never overlap the edge.
3. Cursors should be 32x32 and centered.

Although the cursor does not reach the edge, it does so in a symmetrical manner. The only other limitation is as follows: When running in interlaced modes such as 1080i and 1080p (because it is 1080i and de-interlaced by the HDMI), the cursor is double the size.

Example: This code loops a video and waits for a mouse click or touch screen input. It outputs the coordinates of the click or touch to the shell if it is located 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:
```

Example: This code includes 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: This code includes 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, img)
t.SetRollOverOrigin(1, 0, 0)
```

FILE AND STORAGE OBJECTS

roBrightPackage

An *roBrightPackage* object 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).

Interfaces: *ifBrightPackage*

Object creation:

`CreateObject("roBrightPackage", filename As String)`

- `Filename`: The filename for the zip file.

The *ifBrightPackage* interface provides the following:

- `Unpack(path As String) As Void`: Provides the destination path for the extracted files, e.g. “SD:/”.
- `SetPassword(password As String) As Void`: Provides the password specified when the zip file was created. *roBrightPackage* supports AES 128 and 256 bit encryption as generated by WinZip.
- `GetFailureReason() As String`
- `UnpackFile(a As String, b As String) As Boolean`

Note: ifBrightPackage *is a legacy interface. We recommend you use* roAssetPool *instead to achieve better functionality.*

Example:

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

Using roBrightPackage to distribute new content

BrightSign checks storage devices for autorun scripts in the following order:

1. External USB devices 1 through 9
2. SD
3. µSD

In addition to looking for autorun.brs scripts, BrightSign players look for autorun.zip files that contain the script name autozip.brs. If autozip.brs is encrypted, then the player 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.brs file is found, and autozip.bas is decrypted, then the player will execute the autozip.brs file.

The autozip.brs file cannot reference any external files, as it is the only file to be automatically uncompressed by a BrightSign player prior to execution. The autozip.brs 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("SD:/")
package = 0

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

while true
      sleep(1000)

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

roHashGenerator

This object provides an API for generating a variety of message digests.

Object Creation: The hash algorithm is specified when creating the *roHashGenerator* object.

```
CreateObject(“roHashGenerator”, algorithim As String)
```

The algorithm parameter can take the following strings:

- `SHA256`
- `SHA384`
- `SHA512`
- `SHA1`
- `MD5`
- `CRC32`

Note: `SHA1`, `MD5`, *and* `CRC32` *are only available on firmware versions 4.4.9 or later.*

Interfaces: *ifHashGenerator*

The *ifHashGenerator* interface provides the following.

- `Hash(obj As Object) As Object`: Hashes the payload, which can be supplied in the form of a string (or any object implementing *ifString*) or an *roByteArray*. The hash is returned as an *roByteArray*.

roReadFile, roCreateFile, roReadWriteFile, roAppendFile

These objects provide file I/O functionality.

Object creation:

`CreateObject("roReadFile", filename As String)`: Creating an *roReadFile* object opens the specified file for reading only. Object creation fails if the file does not exist. *roReadFile* implements *ifStreamSeek* and *ifReadStream*.

`CreateObject("roCreateFile", filename As String)`: Creating an *roCreateFile* object opens an existing file or creates a new file. If the file exists, it is truncated to a size of zero. *roCreateFile* implements *ifStreamSeek*, *ifReadStream*, *ifStreamSend*, and *ifFile*.

`CreateObject("roReadWriteFile", filename As String)`: 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*, *ifReadStream*, *ifStreamSend*, and *ifFile*.

`CreateObject("roAppendFile", filename As String)`: 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*.

Interfaces: *ifReadStream*, *ifStreamSend*, *ifStreamSeek*, *ifFile*

The *ifReadStream* interface provides the following:

- `SetReceiveEol(eol_sequence As String) As Void`: Sets the EOL sequence when reading from the stream.

- `ReadByte() As Integer`: 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.
- `ReadByteIfAvailable() As Integer`: Reads a single byte from the stream if one is available. If no bytes are available, it returns immediately. A return value less than 0 indicates either that the EOF has been reached or no byte is available.
- `ReadLine() As String`: Reads until it finds a complete end of the line sequence. If it fails to find the sequence within 4096 bytes, then it returns the 4096 bytes that are found. No data is discarded in this case.
- `ReadBlock(size As Integer) As String`: Reads the specified number of bytes. The number 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.
- `AtEof() As Boolean`: 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 the following:

- `SetSendEol(eol_sequence As String) As Void`: Sets the EOL sequence when writing to the stream.
- `SendByte(byte As Integer) As Void`: Writes the specified byte to the stream.
- `SendLine(string As String) As Void`: Writes the specified characters to the stream followed by the current EOL sequence.
- `SendBlock(string As String) As Void`: Writes the specified characters to the stream. Any null bytes will terminate the block.
- `Flush()`
- `AsyncFlush()`

The *ifStreamSeek* interface provides the following:

- `SeekAbsolute(offset As Integer) As Void`: Seeks 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.
- `SeekRelative(offset As Integer) As Void`: 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*.
- `SeekToEnd As Void`: Seeks to the end of the file.
- `CurrentPosition As Integer`: Retrieves the current position within the file.

The *ifFile* interface provides the following:

- `Flush As Void`: 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).
- `AsyncFlush As Void`: Ensures that all writes will be written out to the file within a few seconds. Without this call (or destroying the object or calling Flush), some writes will be cached indefinitely in the memory.

roRegistry

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")
```

Interfaces: *ifRegistry*

The *ifRegistry* interface provides the following:

- `GetSectionList As roList`: Returns a list with one entry for each registry section.
- `Delete(section As String) As Boolean`: Deletes the specified section and returns an indication of success.
- `Flush As Boolean`: Flushes the registry out to persistent storage.

roRegistrySection

This object represents a section of the registry, enabling the organization of settings within the registry. It allows the section to be read or written.

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

```
CreateObject("roRegistrySection", section As String)
```

Interfaces: *ifRegistrySection*

The *ifRegistrySection* interface provides the following:

- `Read(key As String) As String`: Reads and returns the value of the specified key.
- `Write(key As String, value As String) As Boolean`: Replaces the value of the specified key.
- `Delete(key As String) As Boolean`: Deletes the specified key.
- `Exists(key As String) As Boolean`: Returns True if the specified key exists.
- `Flush As Boolean`: Flushes the contents of the registry out to persistent storage.
- `GetKeyList As roList`: 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 effect immediately to prevent the system from exceeding the maximum number of writes on the onboard persistent storage. At most, 60 seconds after a write to the registry, the newly written data will be automatically 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 exiting the application.

roSqliteDatabase

This object is the main SQLite object that “owns” the database. You can produce as many of these objects as you need.

Interfaces: *ifSqliteDatabase*, *ifSetMessagePort*

The *ifSqliteDatabse* interface provides the following:

- `Open(a As String) As Boolean`
- `Create(a As String) As Boolean`
- `Close()`
- `CreateStatement(a As String) As Object`
- `RunBackground(a As String, b As Object) As Integer`
- `SetMemoryLimit(a As Integer)`
- `SetTempDirectory(a As String)`

The *ifSetMessagePort* interface provides the following:

- `SetPort(a As Object)`

roSqliteEvent

This object is returned by a *RunBackground()* operation if you set a message port on the associated *roSqliteDatabase* object.

Interfaces: *ifSqliteEvent*

The *ifSqliteEvent* interface provides the following:

- `GetTransactionId() As Integer`
- `GetSqlResult() As Integer`

roSqliteStatement

This object can only be created by calling *CreateStatement()* on an *roSqliteDatabase* object.

Interfaces: *ifSqliteStatement*

The *ifSqliteStatement* interface provides the following:

- `BindByName(a As Object) As Boolean`
- `BindByOffset(a As Object) As Boolean`
- `BindText(a As Object, b As String) As Boolean`
- `BindInteger(a As Object, b As Integer) As Boolean`
- `Run() As Integer`
- `RunBackground() As Integer`
- `GetData() As Object`
- `Finalise()`

roStorageAttached, roStorageDetached

Interfaces: *ifString*, *ifStringOps*

The *ifString* interface provides the following:

- `GetString() As String`
- `SetString(a As String)`

The *ifStringOps* interface provides the following:

- `SetString(a As String, b As Integer)`
- `AppendString(a As String, b As Integer)`
- `Len() As Integer`
- `GetEntityEncode() As String`
- `Tokenize(a As String) As Object`
- `Trim() As String`
- `ToInt() As Integer`
- `ToFloat() As Float`
- `Left(a As Integer) As String`
- `Right(a As Integer) As String`
- `Mid(a As Integer) As String`
- `Mid(a As Integer, b As Integer) As String`
- `Instr(a As String) As Integer`
- `Instr(a As Integer, b As String) As Integer`

roStorageHotplug

This object can provide events when storage devices appear or disappear. Currently, only external USB devices are supported, and there is no polling for media.

Interfaces: *ifSetMessagePort*

The *ifSetMessagePort* interface provides the following:

- `SetPort(a As Object)`

roStorageInfo

This object is used to report storage device usage information.

Object creation:

`CreateObject("roStorageInfo", path As String)`: Creates an *roStorageInfo* object containing the storage device information for the specified path. The path does not need to extend to the root of the storage device.

Interfaces: *ifStorageInfo*

The *ifStorageInfo* interface provides the following:

Note: *On some filesystems that have a portion of space reserved for the super-user, the following expression may not be true:* `GetUsedInMegabytes + GetFreeInMegabytes == GetSizeInMegabytes`

- `GetFailureReason() As String`: Yields additional useful information if a function return indicates an error.
- `GetBytesPerBlock As Integer`: Returns the size of a native block on the filesystem used by the specified storage device.
- `GetSizeInMegabytes As Integer`: Returns the total size of the storage device in Mibibytes.
- `GetUsedInMegabytes As Integer`: Returns the amount of space currently used on the storage device in Mibibytes.
  Note: *This amount includes the size of the pool because this class does not integrate pools into its calculations.*
- `GetFreeInMegabytes As Integer`: Returns the available space on the storage device in Mibibytes.
- `GetFileSystemType As String`: Returns a string describing the type of filesystem used on the specified storage. Potential values are *fat12*, *fat16*, *fat32*, *ext3*, *ntfs*, *hfs*, *Hfsplus*.
- `GetStorageCardInfo As Object`: Returns an associative array containing details of the storage device hardware (a memory card, for example). For SD cards, the returned data may include the following:

sd_mfr_id	Int	Card manufacturer ID as assigned by the SD Card Association
sd_oem_id	String	Two-character card OEM identifier as assigned by the SD Card Association
sd_product_name	String	Product name, assigned by the card manufacturer (5 bytes for SD, 6 bytes for MMC)
sd_spec_vers	Int	Version of SD spec to which the card conforms
sd_product_rev	String	Product revision assigned by the card manufacturer
sd_speed_class	String	Speed class (if any) declared by the card
sd_au_size	Int	Size of the SD AU in bytes.

Example:

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

NETWORKING OBJECTS

roAssetCollection

This object is used to represent a collection of assets. This collection can be created by calls to *AddAssets* or *AddAsset*, or by calls to *roSyncPool.GetAssets* ("download" or similar).

Interfaces: *ifAssetCollection, ifInternalAssetCollection*

The *ifAssetCollection* interface provides the following:

- `GetFailureReason() As String`
- `AddAsset(asset_info as Object) As Boolean:` Adds a single asset from an associative array.
- `AddAssets(asset_info_array as Object) As Boolean`: Adds multiple assets from an enumerable object (*roList* or *roArray*) that contains compatible associative arrays.

The associative array contains the following:

`name`	String	Mandatory	The name of the asset. For a file to be realized, it must have a valid filename (i.e. no slashes).
`link`	String	Mandatory	The download location of the asset
`size`	Integer/String	Optional	The size of the asset. Use a string if you want to specify a number that is too large to fit into an integer (this allows file sizes larger than 2 GB).
`hash`	String	Optional	A string in the form of "`hash_algorithm:hash`". See the next table for details.
`change_hint`	String	Optional	Any string that will change in conjunction with the file contents. This is not necessary if the `link` or `hash` is supplied and always changes.

`auth_inherit`	Boolean	Optional	Indication of whether or not this asset uses *roAssetFetcher* authentication information. The default is set to True.
`auth_user`	Boolean	Optional	User to utilize for authentication when downloading only this asset. This automatically disables “`auth_inherit`”.
`auth_passowrd`	Boolean	Optional	Password to use when downloading only this asset. This automatically disables “`auth_inherit`”.
`headers_inherit`	Boolean	Optional	The command to pass any header supplier to *roAssetFetcher* when fetching this asset. The default is true.

Hash algorithms:

`sha1`	If a `sha1` is available, you can validate the hash as the file is downloaded. If such a hash is available, it should be used. The `link` and `change_hint` properties have no effect on the pool file name, so the file is shared even if it is downloaded from different locations.
`besha1`	This algorithm hashes some of the file along with the file size in order to verify the contents. It also moves the `link` and `change_hint` properties into the pool filename.
(none)	Without any hash, the file cannot be verified as it is downloaded, and the system will rely on the `link` and `change_hint` properties to give the pool a unique filename.

roAssetFetcher

Interfaces: *ifAssetFetcher*, *ifMessagePort*, *ifUserData*

The *ifAssetFetcher* interface provides the following:

- `GetFailureReason() As String`
- `EnableUnsafeAuthentication(a As Boolean) As Boolean`
- `AsyncDownload(a As Object) As Boolean`
- `AsyncSuggestCache(a As Object) As Boolean`
- `AsyncCancel() As Boolean`
- `EnablePeerVerification(a As Boolean)`
- `EnableHostVerification(a As Boolean)`
- `SetCertificatesFile(a As String)`
- `SetUserAndPassword(a As String, b As String) As Boolean`
- `AddHeader(a As String, b As String)`
- `SetHeaders(a As Object) As Boolean`
- `SetMinimumTransferRate(a As Integer, b As Integer) As Boolean`
- `SetProxy(a As String) As Boolean`
- `SetFileProgressIntervalSeconds(a As Integer) As Boolean`
- `SetFileRetryCount(a As Integer) As Boolean`
- `SetRelativeLinkPrefix(a As String) As Boolean`
- `BindToInterface(a As Integer) As Boolean`

The *ifMessagePort* interface provides the following:

- `SetPort(a As Object)`

The *ifUserData* interface provides the following:

- `SetUserData(a As Object)`
- `GetUserData() As Object`

roAssetFetcherEvent

Interfaces: *ifAssetFetcherEvent*, *ifUserData*

The *ifAssetFetcherEvent* interface provides the following:

- `GetEvent() As Integer`
- `GetName() As String`
- `GetResponseCode() As Integer`
- `GetFailureReason() As String`
- `GetFileIndex() As Integer`

The *ifUserData* interface provides the following:

- `SetUserData(a As Object)`
- `GetUserData() As Object`

roAssetFetcherProgressEvent

Interfaces: *ifAssetFetcherProgressEvent*, *ifUserData*

The *ifAssetFetcherProgressEvent* interface provides the following:

- `GetFileName() As String`
- `GetFileIndex() As Integer`
- `GetFileCount() As Integer`
- `GetCurrentFileTransferredMegabytes() As Integer`
- `GetCurrentFileSizeMegabytes() As Integer`
- `GetCurrentFilePercentage() As Float`

The *ifUserData* interface provides the following:

- `SetUserData(a As Object)`
- `GetUserData() As Object`

roAssetPool

An instance of *roAssetPool* represents a pool of files for synchronization. You can instruct this object to populate the pool based on a sync spec and then realize it in a specified directory when required.

Object Creation: *roAssetPool* is created with a single parameter representing the rooted path of the pool.

```
CreateObject(“roAssetPool”, pool_path As String)
```

Example:

```
pool = CreateObject (“roAssetPool”, “SD:/pool”)
```

Interfaces: *ifAssetPool*

The *ifAssetPool* interface provides the following:

- `GetFailureReason() As String`
- `ProtectAssets(a As String, b As Object) As Boolean`
- `UnprotectAssets(a As String) As Boolean`
- `UnprotectAllAssets() As Boolean`
- `ReserveMegabytes(a As Integer) As Boolean`
- `GetPoolSizeInMegabytes() As Integer`
- `Validate(a As Object, b As Object) As Boolean`
- `QueryFiles(a As Object) As Object`
- `AssetsReady(a As Object) As Boolean`

roAssetPoolFiles

Interfaces: *ifAssetPoolFiles*

The *ifAssetPoolFiles* interface provides the following:

- `GetFailureReason() As String`
- `GetPoolFilePath(a As String) As String`
- `GetPoolFileInfo(a As String) As Object`

roAssetRealizer

Interfaces: *ifAssetRealizer*

The *ifAssetRealizer* interface provides the following:

- `GetFailureReason() As String`
- `EstimateRealizedSizeInMegabytes(a As Object) As Integer`
- `Realize(a As Object) As Object`
- `ValidateFiles(a As Object, b As Object) As Object`

roAssetRealizerEvent

Interfaces: *ifAssetRealizerEvent*, *ifUserData*

The *ifAssetRealizerEvent* interface provides the following:

- `GetEvent() As Integer`
- `GetName() As String`
- `GetResponseCode() As Integer`
- `GetFailureReason() As String`
- `GetFileIndex() As Integer`

The *ifUserData* interface provides the following:

- `SetUserData(a As Object)`
- `GetUserData() As Object`

roDatagramSender, roDatagramReceiver, roDatagramSocket, roDatagramEvent

The *roDatagramSender* and *roDatagramReceiver* objects allow for simple sending and receiving of unicast and broadcast UDP packets. The *roDatagramEvent* object can be used to both send and receive UDP packets.

roDatagramSender

This object allows UDP packets to be sent to a specified destination.

Object Creation: *roDatagramSender* is created with no parameters:

```
CreateObject("roDatagramSender")
```

Interfaces: *ifDatagramSender*

The *ifDatagramSender* interface provides the following:

- `SetDestination(destination_address As String, destination_port As Integer) As Boolean`: Specifies the destination IP address in dotted quad form along with the destination port. This function returns True if successful.
- `Send(packet As Object) As Integer`: Sends the specified data packet as a datagram. The packet may be a string or an roByteArray. This method returns 0 upon success and a negative error code upon failure.

This example 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”)
```

roDatagramReceiver

This object causes instances of *roDatagramEvent* to be sent to a message port when UDP packets are received on a specified port.

Object Creation: *roDatagramReceiver* is created with a single parameter:

```
CreateObject("roDatagramReceiver ", port As Integer)
```

The `port` paremeter specifies the port on which to receive UDP packets.

Interfaces: *ifIdentity*, *ifSetMessagePort*

The *ifIdentity* interface provides the following:

- `GetIdentity() As Integer`

The *ifSetMessagePort* interface provides the following:

- `SetPort(a As Object)`

This example 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
```

roDatagramSocket

This object both sends and receives UDP packets. Use *roDatagramSocket* if you need the player to communicate using protocols such as SSDP, which only allow a server to respond to the source of a received request.

Received packets are delivered to the message port as *roDatagramEvent* objects.

Interfaces: *ifUserData*, *ifMessagePort*, *ifDatagramSocket*, *ifIdentity*

The *ifUserData* interface provides the following:

- `SetUserData(a As Object)`
- `GetUserData() As Object`

The *ifMessagePort* interface provides the following:

- `SetPort(a As Object)`

The *ifDatagramSocket* interface provides the following:

- `GetFailureReason() As String`: Returns additional information if the `BindToLocalPort` or `Sendto` methods fail.
- `BindToLocalPort(port As Integer) As Boolean`: Binds the socket to the specified local port. Use this method to receive packets sent to a specific port. Alternatively, if you want to receive replies to sent packets (and it doesn't matter which local port is used), pass a port number of 0, and the player will select an unused port. This method returns True upon success and False upon failure
- `GetLocalPort() As Integer`: Returns the local port to which the socket is bound. Use this method if you passed a port number of 0 to `BindToLocalPort` and need to determine which port the player has selected.

- `SendTo(destination_address As String, destination_port As Integer, packet As Object) As Integer`: Sends a single UDP packet, which can be an *roString* or *roByteArray*, to the specified address and port. This method returns 0 upon success and a negative error code upon failure.
- `JoinMulticastGroup(address as String) as Boolean`: Joins the multicast group for the specified address on all interfaces that are currently up. This method returns True upon success and False upon failure. In the event of failure, `GetFailureReason()` may provide additional information. To ensure that you are joined on all network interfaces, you should register for *roNetworkHotplug* events and call the `JoinMulticastGroup()` method in response to the arrival of new networks.

The *ifIdentity* interface provides the following:

- `GetIdentity() As Integer`

roDatagramEvent

Interfaces: *ifUserData*, *ifSourceIdentity*, *ifString*, *ifDatagramEvent*

The *ifUserData* interface provides the following:

- `SetUserData(a As Object)`
- `GetUserData() As Object`

The *ifSourceIdentity* interface provides the following:

- `GetSourceIdentity() As Integer`

The *ifString* interface provides the following:

- `GetString() As String`

The *ifDatagramEvent* interface provides the following:

- `GetByteArray as Object`: Returns the contents of the packet as an *roByteArray*.
- `GetSourceHost as String`: Returns the source IP address of the packet in dotted form.
- `GetSourcePort as Integer`: Returns the source port of the packet.

roHttpEvent

Interfaces: *ifHttpEvent*, *ifUserData*

The *ifHttpEvent* interface provides the following:

- `GetFailureReason() As String`: Yields additional useful information if a function return indicates an error.
- `SetResponseBodyString(a As String)`: Sets the response body for an event generated using the following function: `roHttpServerAddGetFromEvent`. Otherwise, this call is ignored.
- `SetResponseBodyFile(a As String) As Boolean`: Sets the name of a file to use as the source response body for an event generated via `roHttpServer.AddGetFromEvent`. Otherwise, this call is ignored. This function will return False if the file cannot be opened or another failure occurs.

Note: *The file is read gradually as it is sent to the client.*

- `GetRequestBodyString() As String`: Returns the string received if the event was generated via `roHttpServer.AddPostToString`. Otherwise, an empty string is returned.
- `GetRequestHeader(a As String) As String`: Returns the value of the specified HTTP request header. If the header does not exist, an empty string is returned.
- `GetRequestHeaders() As Object`: Returns an *roAssociativeArray* containing all the HTTP request headers.
- `GetRequestParam(a As String) As String`: Returns the value of the specified URI parameter. If the parameter does not exist, an empty string is returned.
- `GetRequestParams() As Object`: Returns an *roAssociativeArray* containing all the URI parameters.
- `AddResponseHeader(a As String, b As String) As Boolean`: Adds the specified HTTP header and value to the response. This function returns True upon success.
- `AddResponseHeaders(a As Object) As Boolean`. Expects to be passed an associative array of header names mapped to header values, which can be of type *roString*, *roInt*, or *roFloat*. Any other value types will cause the request to fail and a subset of headers to possibly be set. This function returns True upon success.

- `GetRequestBodyFile() As String`: Returns the name of the temporary file created if the event is generated via `roHttpServer.AddGetFromEvent`. Otherwise, this call is ignored.
- `SendResponse(a As Integer) As Boolean`: Sends the HTTP response using the specified HTTP status code. To ensure that the response is sent, this function needs be called once you have finished handling the event.
- `GetUrl() As String`
- `GetFormData() As Object`: Returns an *roAssociativeArray* containing all the form data. See roHttpServer.AddPostToFormData for more information.

The *ifUserData* interface provides the following:

- `SetUserData(a As Object)`
- `GetUserData() As Object`

roHttpServer

Interfaces: *ifHttpServer*, *ifSetMessagePort*, *ifGetMessagePort*

The *ifHttpServer* interface provides the following:

- `GetFailureReason() As String`: Yields additional useful information if an *roHttpServer* method fails.
- `AddGetFromFile(a As Object) As Boolean`: Causes any HTTP GET requests for the specified URL path to be met directly from the specified file. You should always specify the MIME type (and possibly the character set) if you expect the request to come from a web browser.
- `AddGetFromEvent(a As Object) As Boolean`: Requests that an *roHttpEvent* event be sent to the configured message port. This occurs when an HTTP GET request is made for the specified URL path.
- `AddPostToString(a As Object) As Boolean`: Requests that an *roHttpEvent* event be sent to the configured message port. This occurs when an HTTP POST request is made for the specified URL path. Use `roHttpEvent.GetRequestBodyString()`to retrieve the posted body.
- `AddPostToFile(a As Object) As Boolean`: Requests that, when an HTTP POST request is made to the specified URL path, the request body will be stored in a temporary file according to the following parameters: "destination_directory". When this request is complete, an *roHttpEvent* event is sent to the configured message port. Use `roHttpEvent.GetRequestBodyFile()` to retrieve the name of the temporary file. If the file still exists at the time the response is sent, it will be automatically deleted.
- `AddPostToFormData(a As Object) As Boolean`: Requests that, when an HTTP POST request is made to the specified URL path, an attempt be made to store form data (passed as `application/x-www-form-urlencoded or multipart/form-data`) in an associative array that can be retrieved by calling `roHttpEvent.GetFormData()`.

The *ifSetMessagePort* interface provides the following:

- `SetPort(a As Object)`

The *ifGetMessagePort* interface provides the following:

- `GetPort() As Object`

roMimeStream

This object passes an MJPEG stream in MIME format to the *roVideoPlayer.PlayFile()* method. There are some limitations to what MJPEG streams this object will play correctly. *roMimeStream* has been optimized to play streaming video from a local source with the smallest possible delay. The result is a short buffering window that is not appropriate for playing MJPEG streams from URLs outside of a local network. We are currently optimizing *roMimeStream* to work with different IP camera brands: see the IP Camera FAQ for more details.

Object Creation: To play an RTSP stream, first instantiate an *roUrlTransfer* object. Then wrap it in an *roMimeStream* object and pass the `PictureStream` to `PlayFile` as follows:

```
u=createobject("roUrlTransfer")
u.seturl("http://mycamera/video.mjpg")
r=createobject("roMimeStream", u)
p=createobject("roVideoPlayer")
p.PlayFile({ PictureStream: r })
```

Interfaces: *ifPictureStream*, *ifMessagePort*

The *ifPictureStream* interface provides the following:

- `GetUrl() As String`

The *ifMessagePort* interface provides the following:

- `SetPort(a As Object)` Posts event messages to the attached message port. The event messages are of the type *roMimeStreamEvent* and will implement the *ifInt* interface. There are currently two possible event messages:
    - `PICTURE_STREAM_FIRST_PICTURE_AVAILABLE = 0`: The first picture is now available for decoding.
    - `PICTURE_STREAM_CONNECT_FAILED`: The object is unable to connect to the specified URL.

roMimeStreamEvent

This object will return an integer corresponding to the event that has occurred:

Interfaces: *ifInt*

The *ifInt* interface provides the following:

- `GetInt() As Integer`
- `SetInt(a As Integer)`

roNetworkAdvertisement

This object is used to advertise services running on a BrightSign player to other devices on the network. The current implementation supports advertising via mDNS (which is part of Zeroconf via Bonjour).

Object creation:

```
CreateObject(“roNetworkAdvertisement”, advertisement as Object) as Object
```

The advertisement should be provided as an *roAssociativeArray* containing the following keys:

- `name`: The service name. This should be a readable string such as “Remote BrightSign Widget Service.”
- `type`: The service type. This should be a service from the definitive list, formatted in the following manner: “_service._protocol” (for example, “_http._tcp”).
- `Port`: The port number on which the service runs.
- `_name`: Any arbitrary text key preceded by an underscore to avoid conflicts within the *roAssociativeArray*.
  Note: *The underscore is removed before the record is registered with mDNS*.

Once the object is created, advertising starts immediately and continues until the object is destroyed (i.e. when it becomes unreferenced).

Interfaces: None

roNetworkAttached, roNetworkDetached

roNetworkAttached

This object implements *ifInt* to report the index of the attached network interface. Instances of this object are posted by *roNetworkHotplug* when a configured network connection becomes available.
Note: *It may take some time after the cable is inserted for this to take place.*

Interfaces: *ifInt*, *ifIntOps*

The *ifInt* interface provides the following:

- `GetInt() As Integer`
- `SetInt(a As Integer)`

The *ifIntOps* interface provides the following:

- `ToStr() As String`

roNetworkDetached

This object implements *ifInt* to report an index of the detached network interface. Instances of this object are posted by *roNetworkHotplug* when a configured network connection becomes unavailable.

The interfaces and methods for *roNetworkDetached* are identical to those outlined for *roNetworkAttached* above.

roNetworkConfiguration

Object creation:

```
CreateObject("roNetworkConfiguration", network_interface as Integer)
```

The *network_interface* parameter is used to distinguish between the following:

- 0 for the Ethernet port (if available) on the rear of the BrightSign.
- 1 for the optional internal Wi-Fi.

Some of the settings are specific to the network interface and some are used by the BrightSign host for all network interfaces.

Interfaces: *ifNetworkConfiguration*

The *ifNetworkConfiguration* interface provides the following:

Note: *"Set" methods do not take effect until "Apply" is called.*

- `SetupDWS(a As Object) As Boolean`
- `GetClientIdentifier() As String`
- `GetProxy() As String`
- `SetClientIdentifier(a As String) As Boolean`
- `SetObfuscatedWiFiPassphrase(a As String) As Boolean`
- `SetInboundShaperRate(a As Integer) As Boolean`

- `SetRoutingMetric(a As Integer) As Boolean`: Configures the metric for the default gateway on the current network interface. Routes with lower metrics are preferred over routes with higher metrics. This function returns True upon success.
- `SetDHCP as Boolean` (interface): Enables DHCP and disables all other settings. This function returns True if successful.
- `SetIP4Address(ip As String) As Boolean` (interface)
- `SetIP4Netmask(netmask As String) As Boolean` (interface)
- `SetIP4Broadcast(broadcast As String) As Boolean` (interface)
- `SetIP4Gateway(gateway As String) As Boolean` (interface): Sets the IPv4 interface configuration. All values must be specified explicitly. Unlike the *ifconfig* shell command, there is no automatic inference. The parameter is a string dotted decimal quad (i.e. “192.168.1.2” or similar). It returns True upon 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")
```

- `SetWiFiESSID(essid as String) as Boolean` (interface): Configures the name of the wireless network to connect to. It returns True on success.
- `SetWiFiPassphrase(passphrase as String) as Boolean`: Configures the passphrase or key for the wireless network. It returns True if successfully set.
- `SetDomain(domain As String) As Boolean` (host): Sets the device domain name. This will be appended to names to fully qualify them, though it is not necessary to call this. This method returns True on success.

Example:

```
nc.SetDomain("brightsign.biz")
```

- `AddDNSServer(server As String)` (host): Adds another server to the list when the object is created and there are no DNS servers. There is currently a maximum of three servers, but adding more will not cause any errors. This method returns True on success. There is no way to remove all the servers; it will be easier to recreate the object instead.
- `GetFailureReason As String`: Returns additional information when a member function returns False.
- `Apply As Boolean`: Applies the requested changes to the network interface. This may take several seconds to complete.
- `SetTimeServer(time_server As String) As Boolean` (host): Sets the default time server, which is "time.brightsignnetwork.com". You can disable the use of NTP by calling *SetTimeServer("")*.
- `GetTimeServer As String` (host): Retrieves the time server currently in use.
- `SetHostName(name as String) as Boolean` (host): Sets the device host name. If no host name has been explicitly set, then a host name is automatically generated based on the device serial number.
- `GetHostName As String` (host): Retrieves the host name currently in use.
- `SetProxy(proxy as String) As Boolean` (host): Sets the name or address of the proxy server used for HTTP and FTP requests. This should be in the form of "http://user:password@hostname:port". It can contain up to four "*" characters, which are each replaced with one octet from the current IP address. For example, if the IP address is currently 192.168.1.2, and the proxy is set to "proxy-*-*", then the BrightSign will attempt to use a proxy named "proxy-192.168".
- `ScanWiFi As Object`: Scans for available wireless networks. The results are reported as an *roArray* containing one or more associative arrays with the following members:

essid	String	Network name
bssid	String	Access point BSSID
signal	Integer	Received signal strength indication. The absolute value of this field is not usually relevant, but it can be compared with the reported value on other networks or in different locations.

- `GetCurrentConfig As Object`: Retrieves the entire current configuration as an associative array containing the following members:

metric	Integer	Interface	Returns the current routing metric for the interface. See `SetRoutingMetric` for more details.
dhcp	Boolean	Interface	Returns True if the system is currently configured to use DHCP. Returns False otherwise.
hostname	String	Host	The currently configured host name
mdns_hostname	String	Host	The Zeroconf host name currently in use. This may be longer than the host name if there is a collision on the current network.
ethernet_mac	String	Interface	The Ethernet MAC address
ip4_address	String	Interface	The current IPv4 address. If none is currently set, the string will be empty.
ip4_netmask	String	Interface	The current IPv4 network mask. If none is currently set, the string will be empty.
ip4_broadcast	String	Interface	The current IPv4 broadcast address. If none is currently set, the string will be empty.
ip4_gateway	String	Interface	The current IPv4 gateway address. If none is currently set, the string will be empty.
domain	String	Host	The current domain suffix
dns_servers	roArray	Host	The currently active DNS servers

	of Strings		
time_server	String	Host	The current time server
configured_proxy	String	Host	The currently configured proxy. This may contain magic characters as explained under SetProxy above.
current_proxy	String	Host	The currently active proxy. Any magic characters will have been replaced as explained under SetProxy above.
type	String	Interface	Either “wired” or “wifi”
link	Boolean	Interface	Indicates whether the network interface is currently connected.
wifi_essid	String	Interface	The name of the current Wi-Fi network (if any)
wifi_signal	Integer	Interface	An indication of the received signal strength. The absolute value of this field is usually not meaningful, but it can be compared with the reported value on other networks or in different locations.

- `TestInterface As Object`: Performs various tests on the network interface to determine whether it appears to be working correctly. It reports the results via an associative array containing the following members:

ok	Boolean	True if the tests find no problems. False if at least one problem was identified.
diagnosis	String	A single-line diagnosis of the first problem identified in the network interface.
log	roArray containing Strings	A complete log of all the tests performed and their results.

- `TestInternetConnectivity As Object`: Performs various tests on the Internet connection (via any available network interface, not necessarily the one specified when the *roNetworkConfiguration* object was created) to determine whether it appears to be working correctly. It reports the results via an associative array containing the following members:

ok	Boolean	True if the tests find no problems. False if at least one problem was identified.
diagnosis	String	A single line diagnosis of the first problem identified with the Internet connection.
log	roArray containing Strings	A complete log of all the tests performed and their results.

roNetworkHotplug

This object can be used to generate events when a network interface becomes available or unavailable. It will post events of the type *roNetworkAttached* and *roNetworkDetached* to the associated message port.
Note: *Reconfiguring a network interface using roNetworkConfiguration may cause it to detach and attach again.*

Interfaces: *ifSetMessagePort*

The *ifSetMessagePort* interface provides the following:

- `SetPort(a As Object)`

roRssParser, roRssArticle

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

roRssParser is created with no parameters:

```
CreateObject("roRssParser")
```

Interfaces: *ifRssParser*, *ifRssArticle*

roRssParser uses the *ifRssParser* interface, which provides the following:

- `ParseFile(filename As String) As Boolean`: Parses an RSS feed from a file.
- `ParseString(filename As String) As Boolean`: Parses an RSS feed from a string.
- `GetNextArticle As Object`: Gets the next article parsed by the RSS parser. The articles are sorted by publication date, with the most recent article first. This returns a *roRssArticle* object if there is one. Otherwise, an integer is returned.

roRssArticle uses the *ifRssArticle* interface, which provides the following:

- `GetTitle As String`: Returns the title of the RSS item.
- `GetDescription As String`: Returns the content of the RSS item.
- `GetTimestampInSeconds`: Returns in seconds the difference in publication date between this RSS item and the most recent item in the feed. The user can utilize this to decide if an article is too old to display.
- `SetTitle(a As String) As Boolean`
- `SetDescription(a As String) As Boolean`
- `SetTimestampInSeconds(a As Integer) As Boolean`

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
```

roRtspStream

This is a simple object that is passed to the *roVideoPlayer.PlayFile()* method. There are some limitations to the RTSP streams this object will play correctly. *roRtspStream* has been optimized to play streaming video from a local source with the smallest possible delay. The result is a short buffering window that is not appropriate for playing RTSP streams from URLs outside of a local network. We are currently optimizing *roRtspStream* to work with different IP camera brands: see the IP Camera FAQ for more details.

Object Creation: To play an RTSP stream, instantiate an *roRtspStream* object with a URL as its argument. Then pass it to the `playfile()` method as follows:

```
r=createobject("roRtspStream", "rtsp://172.30.1.194/axis-media/media.amp")
p=createobject("roVideoPlayer")
p.PlayFile({Rtsp: r})
```

Interfaces: *ifRtspStream*, *ifMessagePort*

The *ifRtspStream* interface provides the following:

- `GetUrl() As String`

The *ifMessagePort* interface provides the following:

- `SetPort(a As Object)`: Posts event messages to the attached message port. The event messages are of the type *roRtspStreamEvent* and will implement the *ifInt* interface.

roRtspStreamEvent

This object will return an integer corresponding to the event that has occurred:

Interfaces: *ifInt*

The *ifInt* interface provides the following:

- `GetInt() As Integer`
- `SetInt(a As Integer)`

roShoutcastStream

Object creation: *roShoutcastStream* takes a URL object, a maximum buffer size (in seconds), and an initial buffering duration (in seconds).

```
CreateObject("roShoutcastStream", url_transfer, buffer size, buffer duration)
```

Interfaces: *ifShoutcastStream*, *ifSetMessagePort*, *ifSourceIdentity*

The *ifShoutcastStream* interface provides the following:

- `GetUrl() As String`
- `GetBufferedDuration() As Integer`
- `GetTimeSinceLastData() As Integer`
- `GetCurrentMetadata() As String`
- `Rebuffer() As Boolean`
- `AsyncSaveBuffer(a As String) As Boolean`
- `RestartBufferRecord() As Boolean`

The *ifSetMessagePort* interface provides the following:

- `SetPort(a As Object)`

The *ifSourceIdentity* interface provides the following:

- `GetSourceIdentity() As Integer`
- `SetSourceIdentity(a As Integer)`

roShoutcastStreamEvent

Interfaces: *ifInt*, *ifSourceIdentity*

The *ifInt* interface provides the following:

- `GetInt() As Integer`
- `SetInt(a As Integer)`

The *ifSourceIdentity* interface provides the following:

- `GetSourceIdentity() As Integer`
- `SetSourceIdentity(a As Integer)`

roSnmpAgent

When this object is created, it starts an SNMP process that handles some standard SNMP MIBs such as system uptime. Prior to starting the *roSnmpAgent*, you can register other OIDs for handling. You can set and retrieve these both by an SNMP client and by the script.

OID values are retrieved by an SNMP client without script interaction, and setting these values simply generates an event from *roSnmpAgent* stating that it has been changed. The script event handler can then retrieve new values and take appropriate action.

Interfaces: *ifSnmpAgent*, *ifSetMessagePort*

The *ifSnmpAgent* interface provides the following:

- `AddOidHandler(a As String, b As Boolean, c As Object) As Boolean`
- `GetOidValue(a As String) As Object`
- `SetOidValue(a As String, b As Object) As Boolean`
- `Start() As Boolean`

The *ifSetMessagePort* interface provides the following:

- `SetPort(a As Object)`
- `Interface ifGetMessagePort:`
- `GetPort() As Object`

roSnmpEvent

Interfaces: *ifString*, *ifStringOps*

The *ifString* interface provides the following:

- `GetString() As String`
- `SetString(a As String)`

The *ifStringOps* interface provides the following:

- `SetString(a As String, b As Integer)`
- `AppendString(a As String, b As Integer)`
- `Len() As Integer`
- `GetEntityEncode() As String`
- `Tokenize(a As String) As Object`
- `Trim() As String`
- `ToInt() As Integer`
- `ToFloat() As Float`
- `Left(a As Integer) As String`
- `Right(a As Integer) As String`
- `Mid(a As Integer) As String`
- `Mid(a As Integer, b As Integer) As String`
- `Instr(a As String) As Integer`
- `Instr(a As Integer, b As String) As Integer`

roStreamByteEvent

Interfaces: *ifInt*, *ifUserData*

The *ifInt* interface provides the following:

- `GetInt() As Integer`
- `SetInt(a As Integer)`

The *ifUserdata* interface provides the following:

- `SetUserData(a As Object)`
- `GetUserData() As Object`

roStreamConnectResultEvent

The event is posted when a new connection is made to an *roTCPServer* port. The normal response to receiving such an event is to create a new *roTCPStream* object and pass the event to its `AcceptFrom` call.

Interfaces: *ifInt*, *ifUserData*

The *ifInt* interface provides the following:

- `GetInt() As Integer`
- `SetInt(a As Integer)`

The *ifUserData* interface provides the following:

- `SetUserData(a As Object)`
- `GetUserData() As Object`

roStreamEndEvent

Interfaces: *ifInt*, *ifUserData*

The *ifInt* interface provides the following:

- `GetInt() As Integer`
- `SetInt(a As Integer)`

The *ifUserData* interface provides the following:

- `SetUserData(a As Object)`
- `GetUserData() As Object`

roStreamLineEvent

Interfaces: *ifUserData*, *ifString*

The *ifUserData* interface provides the following:

- `SetUserData(a As Object)`
- `GetUserData() As Object`

The *ifString* interface provides the following:

- `GetString() As String`
- `SetString(a As String)`

roSyncSpec

This object represents a parsed sync specification. It allows various parts of the specification to be retrieved through methods.

Interfaces: *ifSyncSpec, ifInternalSyncSpec, ifInstanceFromStream*

The *ifSyncSpec* interface provides the following:

- `GetFailureReason() As String`
- `ReadFromFile(a As String) As Boolean`
- `ReadFromString(a As String) As Boolean`
- `WriteToFile(a As String) As Boolean`
- `WriteToString() As String`
- `GetMetadata(a As String) As Object`
- `GetFileList(a As String) As Object`
- `LookupMetadata(a As String, b As String) As String`
- `GetName() As String`
- `EqualTo(a As Object) As Boolean`
- `GetFile(a As String, b As Integer) As Object`
- `FilterFiles(a As String, b As Object) As Object`
- `FilesEqualTo(a As Object) As Boolean`
- `GetAssets(a As String) As Object`

roTCPConnectEvent

The event is posted when a new connection is made to an *roTCPServer* port. The normal response to receiving such an event is to create a new *roTCPStream* object and pass the event to its `AcceptFrom` call.

Interfaces: *ifUserData, ifInternalGetTCPStream*

The *ifUserData* interface provides the following:

- `SetUserData(a As Object)`
- `GetUserData() As Object`

roTCPServer

Interfaces: *ifTCPServerInstance*, *ifUserData*

The *ifTCPServerInstance* interface provides the following:

- `GetFailureReason() As String`: Yields additional useful information if an *roTCPServer* method fails.
- `SetPort(a As Object)`: Sets the message port that will receive events from an *roTCPServer* instance.
- `BindToPort(a As Integer) As Boolean`: Prepares to accept incoming TCP connections on the specified port. The function returns True upon success.

The *ifUserData* interface provides the following:

- `SetUserData(a As Object)`: Supplies an object that will be provided by every event called by an *roTCPServer* instance.
- `GetUserData() As Object`

roTCPStream

Interfaces: *ifStreamReceive*, *ifUserData*, *ifStreamSend*, *ifTCPStream*

The *ifStreamReceive* interface provides the following:

- `SetLineEventPort(a As Object)`
- `SetByteEventPort(a As Object)`
- `SetReceiveEol(a As String)`
- `SetMatcher(a As Object) As Boolean`

The *ifUserData* interface provides the following:

- `SetUserData(a As Object)`: Supplies an object that will be provided by every event called by an *roTCPStream* instance.
- `GetUserData() As Object`

The *ifStreamSend* interface provides the following:

- `SendByte(a As Integer)`
- `SendLine(a As String)`
- `SendBlock(a As Dynamic)`
- `SetSendEol(a As String)`
- `Flush()`

The *ifTCPStream* interface provides the following:

- `GetFailureReason() As String`: Yields additional useful information if an *roTCPStream* method fails.
- `ConnectTo(a As String, b As Integer) As Boolean`: Connects the stream to the specified host (designated using a dotted quad) and port. The function returns True upon success.

- `Accept(a As Object) As Boolean`: Accepts an incoming connection event. The function returns True upon success.
- `AsyncConnectTo(a As String, b As Integer) As Boolean`: Attempts to connect the stream to the specified host (designated using a dotted quad) and port. The function returns False if this action is immediately impossible (for example, when the specified host is not in the correct format). Otherwise, the function returns True upon success. The connect proceeds in the background, and an *roStreamConnectResultEvent* is posted to the associated message port when the connect attempt succeeds or fails.

roUrlEvent

Interfaces: *ifInt*, *ifUserData*, *ifUrlEvent*, *ifString*, *ifSourceIdentity*

The *ifInt* interface provides the following:

- `GetInt As Integer`: Returns the type of event. The following event types are currently defined: transfer complete (1), transfer started (2). Headers are available for suitable protocols (though this is not currently implemented).

The *ifUserData* interface provides the following:

- `SetUserData(a As Object)`
- `GetUserData() As Object`

The *ifUrlEvent* interface provides the following:

- `GetResponseCode As Integer`: Returns the protocol response code associated with an event. The following codes indicate success:
  - `200`: Successful HTTP transfer
  - `226`: Successful FTP transfer
  - `0`: Successful local file transfer

  For unexpected errors, the return value is negative. There are many possible negative errors from the CURL library, but it is often best to look at the text version by calling `GetFailureReason`.

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

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	Failed quote command
-22	CURLE_HTTP_RETURNED_ERROR	
-23	CURLE_WRITE_ERROR	
-25	CURLE_UPLOAD_FAILED	Failed upload command.
-26	CURLE_READ_ERROR	Could not 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	REST command failed.
-33	CURLE_RANGE_ERROR	RANGE command did not work.
-34	CURLE_HTTP_POST_ERROR	
-35	CURLE_SSL_CONNECT_ERROR	Wrong when connecting with SSL.
-36	CURLE_BAD_DOWNLOAD_RESUME	Could not 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 correctly.
-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	Cannot 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	Could not 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 initialize ENGINE.
-67	CURLE_LOGIN_DENIED	User, password, or similar field was not accepted and login failed .
-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 the following URL_easy_setopt options: CURLOPT_CONV_FROM_NETWORK_FUNCTION CURLOPT_CONV_TO_NETWORK_FUNCTION 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 (this is somewhat generic, so the error message will be important when this occurs).
-80	CURLE_SSL_SHUTDOWN_FAILED	Failed to shut down the SSL connection.

- `GetObject() As Object`
- `GetFailureReason As String`: Returns a description of the failure that occurred.
- `GetSourceIdentity As Integer`: Returns a unique number that can be matched with the value returned by *roUrlTransfer.GetIdentity* to determine where this event came from.
- `GetResponseHeaders As roAssociativeArray`: Returns an associative array containing all the headers returned by the server for appropriate protocols (such as HTTP).

The *ifString* interface provides the following:

- `GetString As String`: Returns 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 65,536 characters.

The *ifSourceIdentity* interface provides the following:

- `GetSourceIdentity As Integer`: Returns a unique number that can be matched with the value returned by *roUrlTransfer.GetIdentity* to determine where this event originated.

roUrlStream

This object allows playback of content from a URL; the current implementation is only designed to work from local NAS storage.

This object is created with an associated *roUrlTransfer* object, as well as a number of other numeric parameters that define buffer size, etc. The *roUrlTransfer* object defines the retrieval URL and is documented separately.

To use the final object to play back content, you must put the object into an associative array with the parameter name “Url”. This array can then be sent to *roVideoPlayer.PlayFile()* for playback.

Interfaces: *ifUrlStream*

The *ifUrlStream* interface provides the following:

- `GetUrl() As String`
- `GetBufferSize() As Integer`
- `GetRewindSize() As Integer`
- `GetMinimumFill() As Integer`

roUrlTransfer

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

This object is created with no parameters:

```
CreateObject("roUrlTransfer")
```

Interfaces: *ifUserData*, *ifIdentity*, *ifSetMessagePort*, *ifGetMessagePort*, *ifUrlTransfer*

The *ifUserData* interface provides the following:

- `SetUserData(a As Object)`
- `GetUserData() As Object`

The *ifIdentity* interface provides the following:

- `GetIdentity As Integer`: Returns a unique number that can be used to identify whether events originated from this object.

The *ifSetMessagePort* interface provides the following:

- `SetPort(port As ifMessagePort) As Void`: Sets the message port to which events will be posted for asynchronous requests.

The *ifGetMessagePort* interface provides the following:

- `GetPort() As Object`

The *ifUrlTransfer* interface provides the following:

- `SetUrl(URL As String) As Boolean`: Sets the URL for the transfer request. This function returns False on failure. Use *GetFailureReason* to learn the reason for the failure.

Note
When using `SetUrl` to retrieve content from local storage, you do not need to specify the full file path: `SetUrl("file:/example.html")`. If the content is located somewhere other than the current storage device, you can specify it within the string itself. For example, you can use the following syntax to retrieve content from a storage device inserted into the USB port when the current device is an SD card: `SetUrl("file:///USB1:/example.html")`.

- `AddHeader(name As String, value As String) As Boolean`: Adds the specified HTTP header. This is only valid for HTTP URLs. This function returns False on failure. Use *GetFailureReason* to learn the reason for the failure.
- `GetToString As String`: Connects to the remote service as specified in the URL and returns the response body as a string. This function cannot return until the exchange is complete, and it may block for a long time. Having a single string return means that much of the information (headers, response codes) has been discarded. If you need this information, you can use *AsyncGetToString* instead.
  Note: *The size of the returned string is limited to 65,536 characters.*
- `GetToFile(filename As String) As Integer`: Connects to the remote service as specified in the URL and writes 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 you need this information, use *AsyncGetToFile* instead.
- `AsyncGetToString As Boolean`: Begins 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.

- `AsyncGetToFile(filename As String) As Boolean`: Begins 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.
- `Head As Object`: Synchronously perform an HTTP HEAD request and return the resulting response code and headers through an *roUrlEvent* object. In the event of catastrophic failure (e.g. an asynchronous operation is already active), a null object is returned.
- `AsyncHead As Boolean`: Begins an HTTP HEAD request asynchronously. Events will be sent to the message port associated with the object. A False return indicates that a request could not be issued and no events will be delivered.
- `PostFromString(request As String) As Integer`: Uses the HTTP POST method to post the supplied string to the current URL and return the response code. Any response body is discarded.
- `PostFromFile(filename As String) As Integer`: Uses the HTTP POST method to post the contents of the file specified to the current URL and then return the response code. Any response body is discarded.
- `AsyncPostFromString(request As String) As Boolean`: Uses 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. A False return indicates that the request could not be issued and no events will be delivered.
- `AsyncPostFromFile(filename As String) As Boolean`: Uses the HTTP POST method to post the contents of the specified file to the current URL. Events of the type *roUrlEvent* will be sent to the message port associated with the object A False return indicates that the request could not be issued and no events will be delivered.
- `SetUserAndPassword(user As String, password As String) As Boolean`: 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.
- `SetMinimumTransferRate(bytes_per_second As Integer, period_in_seconds As Integer) As Boolean`: Causes 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.

- `GetFailureReason As String`: May provide additional information if any of the *roUrlTransfer* functions indicate failure.
- `SetHeaders(a As Object) As Boolean`
- `AsyncGetToObject(a As String) As Boolean`
- `AsyncCancel() As Boolean`
- `EnableUnsafeAuthentication(a As Boolean) As Boolean`: Supports basic HTTP authentication if True. HTTP authentication uses an insecure protocol, which might allow others to easily determine the password. The *roUrlTransfer* object will still prefer the stronger digest HTTP if it is supported by the server. If this method is False (which is the default setting), it will refuse to provide passwords via basic HTTP authentication, and any requests requiring this authentication will fail.
- `EnableResume(a As Boolean) As Boolean`
- `EnablePeerVerification(a As Boolean) As Boolean`
- `EnableHostVerification(a As Boolean) As Boolean`
- `SetCertificatesFile(a As String) As Boolean`
- `EnableEncodings(a As Boolean) As Boolean`
- `SetUserAndPassword(a As String, b As String) As Boolean`
- `Head() As Object`
- `Escape(a As String) As String`
- `Unescape(a As String) As String`
- `GetUrl() As String`
- `SetProxy(a As String) As Boolean`
- `SetTimeout(a As Integer) As Boolean`
- `SetUserAgent(a As String) As Boolean`
- `PutFromString(a As String) As Integer`
- `PutFromFile(a As String) As Integer`

- `AsyncPutFromString(a As String) As Boolean`
- `AsyncPutFromFile(a As String) As Boolean`
- `Delete() As Object`
- `AsyncDelete() As Boolean`
- `ClearHeaders()`
- `AddHeaders(a As Object) As Boolean`
- `AsyncMethod(a As Object) As Boolean`
- `SyncMethod(a As Object) As Object`
- `BindToInterface(a As Integer) As Boolean`

INPUT/OUTPUT OBJECTS

roCecInterface

This object provides access to the HDMI CEC channel.

roCecInterface is created with no parameters:

```
CreateObject("roCecInterface")
```

Interfaces: *IfCecInterface*, *ifSetMessagePort*

The *IfCecInterface* interface provides the following:

- `SendRawMessage(packet As Object) As Void`: Sends a message on the CEC bus. The frame data should be provided as an *roByteArray*, with the destination address in the low 4 bits of the first octet. The high 4 bits of the first octet should be supplied as zero; they will be replaced with the source address.

The *ifSetMessagePort* interface provides the following:

- `SetPort(a As Object)`

If an *roMessagePort* is attached to *roCecInterface*, it will receive events of type *roCecRxFrameEvent* and/or *roCecTxCompleteEvent*.

roCecRxFrameEvent implements *ifCecRxFrameEvent*.

The *ifCecRxFrameEvent* interface provides the following:

- `GetByteArray As Object`: Returns the message data as an *roByteArray*.

roCecTxCompleteEvent implements *ifCecTxCompleteEvent*.

The *ifCecTxCompleteEvent* interface provides the following:

- `GetStatusByte As Integer`

roCecRxFrameEvent, roCecTxCompleteEvent

roCecRxFrameEvent

Interfaces: *ifCecRxFrameEvent*

The *ifCecRxFrameEvent* interface provides the following;

- `GetByteArray() As Object`

roCecTxFrameEvent

Interfaces: *ifCecTxCompleteEvent*

The *ifCecTxCompleteEvent* interface provides the following:

- `GetStatusByte() As Integer`

The currently defined status codes are described below:

0x00	Transmission successful
0x80	Unable to send, CEC hardware powered down
0x81	Internal CEC error
0x82	Unable to send, CEC line jammed
0x83	Arbitration error

0x84	Bit-timing error
0x85	Destination address not acknowledged
0x86	Data byte not acknowledged

roChannelManager

You can use this object to manage RF channel scanning and tuning. The roVideoPlayer method also has channel scanning capabilities.

Object Creation: *roChannelManager* is created with no parameters.

```
CreateObject(“roChannelManager”)
```

Interfaces: *ifUserData*, *ifMessagePort*, *ifSetMessagePort*, *ifChannelManager*,

The *ifUserData* interface provides the following:

- `SetUserData(a As Object)`
- `GetUserData() As Object`

The *ifMessagePort* interface provides the following:

- `SetPort(a As Object)`

The *ifSetMessagePort* interface provides the following:

- `SetPort(a As Object)`

The *ifChannelManager* interface provides both a Synchronous and Asynchronous API:

Synchronous API

- `Scan(parameters As roAssociativeArray) As Boolean`: Performs a channel scan on the RF input for both ATSC and QAM frequencies and builds a channel map based on what it finds. The *roChannelManager* object stores a list of all channels that are obtained using the `CreateChannelDescriptor()` method (described below). The list is cleared on each call to `Scan()` by default, but this behavior can be overridden.

Each channel takes approximately one second to scan; you can limit the scope of the channel scan with the following parameters:

  - `[“ChannelMap”] = ”ATSC”` or `”QAM”`: Limits the frequency scan to either QAM or ATSC.
  - `[“ModulationType”] = ”QAM64”` or `”QAM256”`: Limits the modulation type of the scan to QAM64 or QAM256.
  - `[“FirstRfChannel”] = Integer` and/or `[“LastRfChannel”] = Integer`: Limits the scan to the specified range of channels. The high end of the channel range is an optional parameter.
  - `[“ChannelStore”] = “DISCARD ALL”` or `“MERGE”`: Controls how the script handles previous channel scan information. The default setting is `DISCARD ALL`, which clears all channel data prior to scanning. On the other hand, `MERGE` overwrites the data only for channels specified in the scan.

- `GetChannelCount() As Integer`: Returns the number of found channels.
- `ClearChannelData() As Boolean`: Clears all stored channel scanning data, including that which persists in the registry. This method also cancels any `AsyncScan()` calls that are currently running.
- `GetCurrentSnr() As Integer`: Returns the SNR (in centibels) of the currently tuned channel.
- `ExporttoXML() As String`: Serializes the contents of RF channels into XML. You can write the XML to a file that can be used at a later point on the same or other units. See below for an example of XML output.
- `ImportFromXML(a As String) As Boolean`: Retrieves the RF channel contents stored as XML. The formatting of the XML is controlled using version tags.

Example

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<!DOCTYPE boost_serialization>
<boost_serialization signature="serialization::archive" version="7">
<ChannelList class_id="0" tracking_level="0" version="0">
<ChannelCount>2</ChannelCount>
<Channel class_id="1" tracking_level="0" version="0">
 <RfChannel>42</RfChannel>
 <ModulationType>7</ModulationType>
 <SpectralInversion>0</SpectralInversion>
 <MajorChannelNumber>1</MajorChannelNumber>
 <MinorChannelNumber>1</MinorChannelNumber>
</Channel>
<Channel>
 <RfChannel>42</RfChannel>
 <ModulationType>7</ModulationType>
 <SpectralInversion>0</SpectralInversion>
 <MajorChannelNumber>1</MajorChannelNumber>
 <MinorChannelNumber>2</MinorChannelNumber>
</Channel>
</ChannelList>
```

- `EnableScanDebug(filename As String) As Boolean`: Allows all scan debugging to be written to a text file. By default, there is no debug output from a scan. You can close the debug file by passing an empty string.

  Example

```
c=CreateObject("roChannelManager")
c.EnableScanDebug("tmp:/scandebug.txt")

v = CreateObject("roVideoPlayer")
aa = CreateObject("roAssociativeArray")
aa["RfChannel"] = 12
aa["VirtualChannel"] = "24.1"
print v.PlayFile(aa)
```

```

c.EnableScanDebug("")
```

- `CreateChannelDescriptor(a As Object) As Object`: Creates an associative array that can either be passed to the *roVideoPlayer.PlayFile* method (to tune to a channel) or parsed for metadata. The generated channel object can be based on one of the following:
  - Index:

    ```
    [“ChannelIndex”] = 0
    ```

  - Virtual channel number as a string in an associative array:

    ```
    [“VirtualChannel”] = ”12.1”
    ```

  - Channel name as a string:

    ```
    [“ChannelName”] = ”KCBS”
    ```

  Note: *Channels are sorted internally by virtual channel, so you could use a `ChannelIndex` script to implement standard channel up/down behavior.*

  These are the entries generated in the array:
  - `VirtualChannel`
  - `ChannelName`
  - `CentreFrequency`
  - `ModulationType`
  - `VideoPid`
  - `VideoCodec`
  - `AudioPid`

- `AudioCodec`
- `SpectralInversion`
- `ChannelMap`
- `FirstRfChannel`
- `LastRfChannel`

The last three entries in this array allow you to use the same *roArray* as a parameter for `Scan()` and `PlayFile()`. The first and last RF channel values are set to the same value so that only one RF channel will be scanned. This kind of scan can be performed at the same time as playing the channel because it doesn't require retuning.

Example

```
c=CreateObject("roChannelManager")
aa=CreateObject("roAssociativeArray")
aa["ChannelMap"] = "QAM"
aa["FirstRfChannel"] = 10
aa["LastRfChannel"] = 15
c.Scan(aa)

cinfo  = CreateObject("roAssociativeArray")
cinfo["ChannelIndex"] = 0
desc = c.CreateChannelDescriptor(cinfo)
print desc

v = CreateObject("roVideoPlayer")
v.PlayFile(desc)
c.Scan(desc)
```

Asynchronous API

- `AsyncScan(parameters As roAssociativeArray) As Boolean`: Begins a channel scan on the RF input and returns the results immediately. Otherwise, the behavior and parameters of this method are identical to `Scan()`. When completed or cancelled, `AsyncScan()` generates an *roChannelManagerEvent*, which supports *ifUserData* and outputs two types of event:
  - 0 – Scan Complete: Generated upon the completion of a scan. No extra data is supplied.
  - 1 – Scan Progress: Generated upon every tune that is performed during the scan. `GetData()` returns the percentage complete of the scan.
- `CancelScan() As Boolean`: Cancels any asynchronous scans that are currently running. This method does not generate an *roChannelManagerEvent*.

Synchronous Example

```
c = CreateObject("roChannelManager")

' Scan the channels
aa  = CreateObject("roAssociativeArray")
aa["ChannelMap"] = "ATSC"
aa["FirstRfChannel"] = 12
aa["LastRfChannel"] = 50
c.Scan(aa)

' Start at the first channel
index = 0
cinfo  = CreateObject("roAssociativeArray")
cinfo["ChannelIndex"] = index
desc = c.CreateChannelDescriptor(cinfo)
```

```
' Play the first channel
v = CreateObject("roVideoPlayer")
v.PlayFile(desc)

' Play the second channel
index = index + 1
cinfo["ChannelIndex"] = index
desc = c.CreateChannelDescriptor(cinfo)
v.PlayFile(desc)
```

Asynchronous Example

```
c = CreateObject("roChannelManager")
p = CreateObject("roMessagePort")
c.SetPort(p)

' Scan the channels
aa  = CreateObject("roAssociativeArray")
aa["ChannelMap"] = "ATSC"
aa["FirstRfChannel"] = 12
aa["LastRfChannel"] = 50
c.AsyncScan(aa)

loop:
  msg = wait(2000,p)
  if msg = 0 then goto scan_complete
  goto loop

scan_complete:
' Start at the first channel
index = 0
cinfo  = CreateObject("roAssociativeArray")
cinfo["ChannelIndex"] = index
desc = c.CreateChannelDescriptor(cinfo)

' Play the first channel
v = CreateObject("roVideoPlayer")
v.PlayFile(desc)

' Rescan the current channel, and update the
desc["ChannelStore"] = MERGE
c.Scan(desc)
```

roControlPort

This object is an improved version of *roGpioControlPort*. It provides support for the IO port of the BP200 and BP900 USB botton boards as well as the on-board IO port and side buttons on the BrightSign player. It also supports button-up events. The object is used to configure output levels on the IO connector and monitor inputs. Typically, LEDs and buttons are attached to the IO connector on the BrightSign player or the BrightSign Expansion Module.

Object creation:

```
CreateObject("roControlPort", name as String)
```

For `name`, use one of the following:

- `BrightSign` to specify the onboard DA-15 connector (on HD210, HD410, HD1010w, HD1020) and side buttons (i.e. SVC button and reset button).
- `Expander-GPIO` to specify the DB-25 connector on the BrightSign Expansion Module. If no BrightSign Expansion module is attached, then object creation will fail and Invalid will be returned.
- `Expander-DIP` to specify the eight DIP switches on the BrightSign Expansion Module. If no BrightSign Expansion module is attached, then object creation will fail and Invalid will be returned.

Note: *Hot-plugging the BrightSign Expansion Module is not supported.*

Interfaces: *ifControlPort*, *ifSetMessagePort*

The *ifControlPort* interface provides the following:

- `GetVersion as String`: Returns the version number of the firmware (either the main BrightSign firmware or the BrightSign Expansion Module firmware) responsible for the control port.

- `EnableOutput(pin as Integer) as Boolean`: Marks the specified pin as an output. If an Invalid pin number is passed, False will be returned. If successful, the function returns True. The pin will be driven high or low depending on the current output state of the pin.
- `EnableInput(pin as Integer) as Boolean`: Marks the specified pin as an input. If an Invalid pin number is passed, False will be returned. If successful, the function returns True. The pin will be tri-stated and can be driven high or low externally.
- `GetWholeState as Integer`: Returns the state of all the inputs attached to the control port as bits in an integer. The individual pins can be checked using binary operations, although it is normally easier to call *IsInputActive* instead.
- `IsInputActive(pin as Integer) as Boolean`: Returns the state of the specified input pin. If the pin is not configured as an input, then the result is undefined.
- `SetWholeState(state as Integer)`: Specifies the desired state of all outputs attached to the control port as bits in an integer. The individual pins can be set using binary operations, although it is normally easier to call *SetOutputState* instead.
- `SetOutputState(pin as Integer, level as Boolean):` Specifies the desired state of the specified output pin. If the pin is not configured as an output, the resulting level is undefined.
- `GetIdentity as Integer`: Returns the identity value that can be used to associate *roControlUp* and *roControlDown* events with this control port.

The *ifSetMessagePort* interface provides the following:

- `SetPort(port as Object)`: Requests that all events raised on this control port be posted to the specified message port.

roControlUp, roControlDown

These objects are posted by the control port to the configured message port when inputs change state. The *roControlUp* and *roControlDown* objects are not normally created directly.

An *roControlDown* event is posted when the input level goes from high to low. An *roControlUp* event is posted when the input level goes from low to high.

Interfaces: *ifInt*, *ifSourceIdentity*

The *ifInt* interface provides the following:

- `GetInt as Integer`: Retrieves the pin number associated with the event.

The *ifSourceIdentity* interface provides the following:

- `GetSourceIdentity as Integer`: Retrieves the identity value that can be used to associate events with the source *roControlPort* instance.

roGpioControlPort, roGpioButton

Note*: New scripts should use roControlPort instead of roGpioControlPort.*

roGpioControlPort

This object is used to control and wait for events on the BrightSign generic DB15 control port. Typically, LEDs or buttons are connected to the DB15 / DB25 port. Turning on a GPIO output changes the voltage on the GPIO port to 3.3V. Turning off a GPIO output changes the voltage on the GPIO port to 0V.

On all BrightSign models, the GPIO ports are bidirectional and must be programmed as either inputs or outputs. The IDs range from 0–7. *SetWholeState* will overwrite any prior output settings. *SetOutputState* takes an output ID (1, 2, or 6 for example). *SetWholeState* takes a mask (for example, *SetWholeState(2^1 + 2^2)* to set IDs 1 and 2).

Interfaces: *ifSetMessagePort*, *ifGpioControlPort*

The *ifSetMessagePort* interface provides the following:

- `SetPort(obj As Object) As Void`

The *ifGpioControlPort* interface provides the following:

- `IsInputActive(input_id As Integer) As Boolean`
- `GetWholeState As Integer`
- `SetOutputState(output_id As Integer, onState As Boolean) As Void`
- `SetWholeState(on_state As Integer) As Void`
- `EnableInput(input_id As Integer) As Boolean (HD410, HD810, HD1010 only)`
- `EnableOutput(output_id As Integer) As Boolean (HD410, HD810, HD1010 only)`

roGpioButton

Interfaces: *ifInt*.

The ifInt interface contains the input ID listed above and provides the following:

- `GetInt As Integer`

roIRRemote

This component supports receiving and transmitting arbitrary Infrared remote control codes using the NEC protocol. Codes are expressed in 24 bits.

- Bits 0-7: Button code
- Bits 8-23: Manufacturer code

If the manufacturer code is zero, then the code is considered to be intended for the Roku SoundBridge remote control.

The best way to determine the values required is to capture the codes received by *roIRRemote* when the remote buttons of the device are pressed and then send the same codes.

Interfaces: *ifSetMessagePort*, *ifIRRemote*.

The *ifSetMessagePort* interface provides the following:

- `SetPort(message_port_object As Object) As Void`

The *ifIRRemote* interface provides the following:

- `Send(protocol as String, code as Integer) As Boolean`

Note: *The only protocol currently supported is "NEC." The return value is True if the code was successfully transmitted, although there is no way to determine if the controlled device actually received it.*

roIRRemotePress

Messages are generated upon key presses from the Roku Soundbridge remote.

Interfaces: *ifInt*, *ifIntOps*

The *ifInt* interface contains keycode and provides the following:

- `GetInt As Integer`
- `SetInt(a As Integer)`

The *ifIntOps* interface provides the following:

- `ToStr() As String`

For the Roku SoundBridge remote control, the Integer 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

roKeyboard, roKeyboardPress

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

Interfaces: *ifSetMessagePort*

The *ifSetMessagePort* interface provides the following:

- `SetPort(As Object) As Void`

roKeyboardPress
This object is a keyboard event resulting from the user pressing a key on a USB keyboard. The *int* value is equivalent to the ASCII code of the key that was pressed.

Interfaces: *ifInt, ifIntOps*

The *ifInt* interface contains the ASCII value of key presses and provides the following:

- `GetInt As Integer`
- `SetInt(a As Integer)`

The *ifIntOps* interface provides the following:

- `ToStr() As String`

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		

roMessagePort

A message port is the destination where messages (events) are sent. See the Event Loops section for more details. You do not call these functions directly when using BrightScript. Instead, use the “Wait” BrightScript statement (see the BrightScript documentation for more details).

Interfaces: *ifMessagePort*, *ifEnum*

The *ifMessagePort* interface provides the following:

- `GetMessage As Object`
- `WaitMessage(timeout As Integer) As Object`
- `PostMessage(msg As Object) As Void`
- `PeekMessage() As Object`

The *ifEnum* interface provides the following:

- `Reset()`: Resets the position to the first element of enumeration.
- `Next() As Dynamic`: Returns the typed value at the current position and increment position.
- `IsNext() As Boolean`: Returns True if there is a next element.
- `IsEmpty() As Boolean`: Returns True if there is not a next element.

roSequenceMatcher

Interfaces: *ifStreamReceiveObserver, ifSequenceMatcher*

The *ifSequenceMatcher* interface provides the following:

- `SetMessagePort(a As Object)`
- `Add(a As Object, b As Object) As Boolean`

roSequenceMatchEvent

Interfaces: *ifUserData*

The *ifUserData* interface provides the following:

- `SetUserData(a As Object)`
- `GetUserData() As Object`

roSerialPort

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

roSerialPort sends the following event types:

- `roStreamLineEvent`: The line event is generated whenever the end of line string set using *SetEol* is found and contains a String for the whole line. This object implements the *ifString* and *ifUserData* interfaces.
- `roStreamByteEvent`: The byte event is generated on every byte received. This object implements the *ifInt* and *ifUserData* interfaces.

Interfaces: *ifStreamSend*, *ifStreamReceive*, *ifSerialControl,* *ifUserData*

The *ifStreamSend* interface provides the following:

- `SendByte(byte As Integer) As Void`
- `SendLine(line As String) As Void`
- `SendBlock(block As String) As Void`
- `SetSendEol(a As String) As Void`
- `Flush()`

The *ifStreamReceive* interface provides the following:

- `SetLineEventPort(port As Object) As Void`
- `SetByteEventPort(port As Object) As Void`
- `SetReceiveEol(a As String)`
- `SetMatcher(a As Object) As Boolean`

The *ifSerialControl* interface provides the following:

- `SetBaudRate(baud_rate As Integer) As Boolean`: Sets the baud rate of the device. The supported baud rates are as follows: 1800, 2000, 2400, 3600, 4800, 7200, 9600, 12800, 14400, 19200, 23040, 28800, 38400, 57600, and 115200.
- `SetMode(mode As String) As Boolean`: Sets the serial mode in “8N1” syntax. The first character is the number of data bits. It can be either 5, 6, 7, or 8. The second number is the parity. It can be “N”one, “O”dd, or “E”ven. The third is the number of stop bits. It can be 1 or 2.
- `SetEcho(enable As Boolean) As Boolean`: Enables or disables serial echo. It returns True on success and False on failure.
- `SetEol(a As String)`
- `SetInverted(a As Boolean) As Boolean`

The *ifUserData* interface provides the following:

- `SetUserData(user_data As Object)`: Sets the user data that will be returned when events are raised.
- `GetUserData() As Object`: Returns the user data that has previously been set via *SetUserData*. It will return Invalid if no data has been set.

Example: This code waits for a serial event and echoes 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.
```

SYSTEM OBJECTS

roDeviceInfo

Interfaces: *ifDeviceInfo*

The *ifDeviceInfo* interface provides the following:

- `GetModel As String`: Returns the model name for the BrightSign device running the script as a string (for example, "HD1010" or "HD110").
- `GetVersion As String`: Returns the version number of the BrightSign firmware running on the device (for example, "4.0.13").
- `GetVersionNumber As Integer`: Returns the version number of the BrightSign firmware running on the device in the more comparable numeric form of (major*65536 + minor*256 + build).
- `GetBootVersion As String`: Returns the version number of the BrightSign boot firmware, also known as "safe mode", as a string (for example, "1.0.4").
- `GetBootVersionNumber As Integer`: 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).
- `GetDeviceUptime As Integer`: Returns the number of seconds that the device has been running since the last power cycle or reboot.
- `GetDeviceUniqueId As String`: Returns an identifier that, if not an empty string, is unique to the unit running the script.
- `GetFamily As String`: Returns a single string that indicates the family to which the device belongs. A device family is a set of models that are all capable of running the same firmware.
- `GetDeviceLifetime() As Integer`

- `HasFeature(a As String) As Boolean`: Returns True if the player feature, passed as a case-insensitive string parameter, is present on the current device and firmware. These are the possible features you can query from the script:

  Note: *If you pass a parameter other than one of those listed below, it may return False even if the feature is available on the hardware and firmware.*

  - `"brightscript1"`: BrightScript Version 1
  - `"brightscript2"`: BrightScript Version 2
  - `"networking"`: Any form of networking capability; there may be no network currently available.
  - `"hdmi"`
  - `"component video"`
  - `"vga"`
  - `"audio1"`: The first audio output
  - `"audio2"`: The second audio output (true on the HD2000 only)
  - `"audio3"`: The third audio output (true on the HD2000 only)
  - `"ethernet"`
  - `"usb"`
  - `"rs232"`
  - `"5v serial"`
  - `"gpio connector"`
  - `"gpio12 button"`
  - `"reset button"`
  - `"rtc"`
  - `"registry"`
  - `"nand storage"`
  - `"sd"`: SD or SDHC
  - `"sdhc"`: SDHC only

Example:

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

On a particular system, this generates:

```
HD1010
3.2.41           197161
3.2.28           197148
 14353            3129
```

roResourceManager

The *roResourceManager* is used for managing strings in multiple languages.

Object creation:

`CreateObject("roResourceManager", filename As String)`: *filename* is 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.

Interfaces: *ifResourceManager*

The interface *ifResourceManager* provides the following:

- `SetLanguage(language_identifier As String) As Boolean`: Instructs the *roResourceManager* object to use the specified language. False is returned if there are no resources associated with the specified language.
- `GetResource(resource_identifier As String) As String`: Returns the resource string in the current language for a given resource identifier.
- `GetFailureReason() As String`: Yields additional useful information if a function return indicates an error.
- `GetLanguage() As String`

At present, *roResourceManager* is primarily used for localizing the *roClockWidget*. The resource file passed in during 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 square brackets is the resource identifier. Each line after it is a language identifier followed by the resource string. Multiple *roResourceManager* objects can be created. A default “resources.txt” file, which contains a range of internationalization for the clock widget, is available from the BrightSign website.

roSystemLog

This object enables the application to receive events that are intended for reporting errors and trends, rather than for triggering a response to a user action.

roSystemLog requires specific design patterns in your BrightScript application:

- Use one *roMessagePort* throughout the application (instead of creating a new *roMessagePort* for each screen).
- Create one *roSystemLog* instance at startup that remains for the entire lifetime of the application.
- Pass the global *roMessagePort* mentioned above to `SetMessagePort()` on the *roSystemLog* component.
- Enable the desired log types using `EnableType()`.

This object is created with no parameters:

```
CreateObject("roSystemLog")
```

Interfaces: *ifStreamSend*, *ifSystemLog*

The *ifStreamSend* interface provides the following:

- `SendByte(a As Integer)`
- `SendLine(a As String)`
- `SendBlock(a As Dynamic)`
- `SetSendEol(a As String)`
- `Flush()`

The *ifSystemLog* interface provides the following:

- `ReadLog() As Object`: Enables log messages of the type `LogType`. When a `LogType` is enabled, the log events are sent to the message port using `SetMessagePort()`. All system log events are disabled by default and must be explicitly enabled by the application. The current valid forms of `LogType` are as follows:
  - `http.error`: Sent whenever an error occurs while executing an HTTP request. This could be during or after the initial connection (i.e. while reading data from the body).
  - `http.connect`: Sent whenever a successful HTTP connection is made, meaning that the server responded to the HTTP request and there were no errors in the headers of the response. This message indicates nothing about the body of the HTTP request.
  - `Bandwidth.minute`: Sent every minute to indicate the current measured bandwidth.

DATE AND TIME OBJECTS

roDateTime

roDateTime represents an instant in time. See *roSystemTime* and *roTimer* for other objects with timing functionality.

Interfaces: *ifDateTime*, *ifString*

The *ifDateTime* interface provides the following:

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

- `AddMilliseconds(milliseconds As Integer) As Void`
- `SubtractMilliseconds(milliseconds As Integer) As Void`
- `Normalize As Boolean`: Checks that all the fields supplied are correct. This function fails if the values are out of bounds.
- `ToIsoString() As String`
- `FromIsoString(a As String) As Boolean`
- `ToSecondsSinceEpoch() As Integer`
- `FromSecondsSinceEpoch(a As Integer) As Boolean`
- `GetString() As String`

The *ifString* interface provides the following:

- `GetString() As String`

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

roSystemTime

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. See *roDateTime* and *roTimer* for other objects with timing functionality.

Interfaces: *ifSystemTime*.

The *ifSystemTime* interface provides the following:

- `GetLocalDateTime As ifDateTime`
- `GetUtcDateTime As ifDateTime`
- `GetZoneDateTime(timezone_name As String) As ifDateTime`
- `SetLocalDateTime(localDateTime As roDateTime) As Boolean`
- `SetUtcDateTime(utcDateTime As roDateTime) As Boolean`
- `GetTimeZone As String`
- `SetTimeZone(zone_name As String) As Boolean`
- `IsValid As Boolean`: Returns True if the system time is set to something valid. It can be set from the RTC or NTP.

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 Savings (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 behind Greenwich Mean Time
GMT-2: 2 hours behind Greenwich Mean Time
GMT-3: 3 hours behind Greenwich Mean Time
GMT-3:30: 3.5 hours behind Greenwich Mean Time
GMT-4: 4 hours behind Greenwich Mean Time
GMT-4:30: 4.5 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-9:30: 9.5 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
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+3:30: 3.5 hours ahead of Greenwich Mean Time

GMT+4: 4 hours ahead of Greenwich Mean Time
GMT+4:30: 4.5 hours ahead of Greenwich Mean Time
GMT+5: 5 hours ahead of Greenwich Mean Time
GMT+5:30: 5.5 hours ahead of Greenwich Mean Time
GMT+6: 6 hours ahead of Greenwich Mean Time
GMT+6:30: 6.5 hours ahead of Greenwich Mean Time
GMT+7: 7 hours ahead of Greenwich Mean Time
GMT+7:30: 7.5 hours ahead of Greenwich Mean Time
GMT+8: 8 hours ahead of Greenwich Mean Time
GMT+8:30: 8.5 hours ahead of Greenwich Mean Time
GMT+9: 9 hours ahead of Greenwich Mean Time
GMT+9:30: 9.5 hours ahead of Greenwich Mean Time
GMT+10: 10 hours ahead of Greenwich Mean Time
GMT+10:30: 10.5 hours ahead of Greenwich Mean Time
GMT+11: 11 hours ahead of Greenwich Mean Time
GMT+11:30: 11.5 hours ahead of Greenwich Mean Time
GMT+12: 12 hours ahead of Greenwich Mean Time
GMT+12:30: 12.5 hours ahead of Greenwich Mean Time
GMT+13: 13 hours ahead of Greenwich Mean Time
GMT+14: 14 hours ahead of Greenwich Mean Time

roTimer

See *roDateTime* and *roSystemTime* for other objects with timing functionality.

Interfaces: *ifTimer*, *ifIdentity*, *ifSetMessagePort*

The *ifTimer* interface provides the following:

- `SetTime(hour As Integer, minute As Integer, second As Integer, millisecond As Integer) As Void`
- `SetDate(year As Integer, month As Integer, day As Integer) As Void`
- `SetDayOfWeek(day_of_week As Integer) As Void`: Sets the time when 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 specifications must be zero. Events will be raised at most once per minute near the whole minute.
- `SetDateTime(As ifDateTime) As Void`: Sets the time when you wish the event to trigger from an *roDateTime* object. It is not possible to set wildcards using this method.
- `Start As Boolean`: Starts the timer based on the current values specified using the above functions.
- `Stop As Boolean`: Stops the timer.
- `SetTime(a As Integer, b As Integer, c As Integer)`

The *ifIdentity* interface provides the following:

- `GetIdentity() As Integer`
- `SetIdentity(a As Integer)`

The *ifSetSetMessagePort* interface provides the following:

- `SetPort(a As Object)`

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
```

roTimerEvent

Interfaces: *ifSourceIdentity*

The *ifSourceIdentity* interface provides the following.

- `GetSourceIdentity() As Integer`
- `SetSourceIdentity(a As Integer)`

roTimeSpan

This object provides an interface to a simple timer for tracking the duration of activities. It is useful for tracking how long an action has taken or whether a specified time has elapsed from a starting event.

Interfaces: *ifTimeSpan*

The *ifTimeSpan* interface provides the following:

- `Mark()`
- `TotalMilliseconds() As Integer`
- `TotalSeconds() As Integer`
- `GetSecondsToISO8601Date(a As String) As Integer`

LEGACY OBJECTS

roSyncPool

We recommend using roAssetPool instead.

Object Creation:

```
CreateObject(“roSyncPool”, pool_path As String)
```

Example:

```
pool = CreateObject (“roSyncPool”, “SD:/pool”)
```

Interfaces: *ifSyncPool*, *ifIdentity*, *ifMessagePort*, *ifUserData*

The *ifSyncPool* interface provides the following:

- `GetFailureReason() As String`
- `AsyncDownload(a As Object) As Boolean`
- `AsyncCancel() As Boolean`
- `Realize(a As Object, b As String) As Object`
- `ProtectFiles(a As Object, b As Integer) As Boolean`
- `ReserveMegabytes(a As Integer) As Boolean`
- `GetPoolSizeInMegabytes() As Integer`
- `EstimateRealizedSizeInMegabytes(a As Object, b As String) As Integer`
- `IsReady(a As Object) As Boolean`

- `Validate(a As Object, b As Object) As Boolean`
- `EnablePeerVerification(a As Boolean)`
- `EnableHostVerification(a As Boolean)`
- `SetCertificatesFile(a As String)`
- `SetUserAndPassword(a As String, b As String) As Boolean`
- `AddHeader(a As String, b As String)`
- `SetHeaders(a As Object) As Boolean`
- `SetMinimumTransferRate(a As Integer, b As Integer) As Boolean`
- `AsyncSuggestCache(a As Object) As Boolean`
- `SetProxy(a As String) As Boolean`
- `ValidateFiles(a As Object, b As String, c As Object) As Object`
- `SetFileProgressIntervalSeconds(a As Integer) As Boolean`
- `QueryFiles(a As Object) As Object`
- `SetFileRetryCount(a As Integer) As Boolean`
- `SetRelativeLinkPrefix(a As String) As Boolean`
- `BindToInterface(a As Integer) As Boolean`
- `EnableUnsafeAuthentication(a As Boolean)`

The *ifIdentity* interface provides the following:

- `GetIdentity() As Integer`

The *ifMessagePort* interface provides the following:

- `SetPort(a As Object)`

The *ifUserData* interface provides the following:

- `SetUserData(a As Object)`
- `GetUserData () As Object`

roSyncPoolEvent

We recommend using *roAssetFetcherEvent* instead.

Interfaces: *ifSourceIdentity*, *ifSyncPoolEvent*, *ifUserData*

The *ifSourceIdentity* interface provides the following:

- `GetSourceIdentity() As Integer`

The *ifSyncPoolEvent* interface provides the following

- `GetEvent() As Integer`
- `GetName() As String`
- `GetResponseCode() As Integer`
- `GetFailureReason() As String`
- `GetFileIndex() As Integer`

The *ifUserData* interface provides the following:

- `SetUserData(a As Object)`
- `GetUserData() As Object`

roSyncPoolFiles

We recommend using *roAssetPoolFiles* instead.

Interfaces: *ifSyncPoolFiles*

The *ifSyncPoolFiles* interface provides the following:

- `GetFailureReason() As String`
- `GetPoolFilePath(a As String) As String`
- `GetPoolFileInfo(a As String) As Object`

roSyncPoolProgressEvent

We recommend using *roAssetFetcherProgressEvent* instead.

Interfaces: *ifSourceIdentity*, *ifSyncPoolProgressEvent*, *ifUserData*

The *ifSourceIdentity* interface provides the following:

- `GetSourceIdentity() As Integer`

The *ifSyncPoolProgressEvent* interface provides the following:

- `GetFileName() As String`
- `GetFileIndex() As Integer`
- `GetFileCount() As Integer`
- `GetCurrentFileTransferredMegabytes() As Integer`
- `GetCurrentFileSizeMegabytes() As Integer`
- `GetCurrentFilePercentage() As Float`

The *ifUserData* interface provides the following:

- `SetUserData(a As Object)`
- `GetUserData() As Object`

CHANGE LOG

4.4.x, 4.2.x, 3.10.x

March 1, 2013

1.1 Added entries for *roDatagramSocket* [implemented in version 4.4.47], *roXMLElement*, *roAudioPlayerMx*, and *roAudioEventMx*.

1.2 Added entry for *roChannelManager* [implemented in version 4.4.51].

1.3 Added entries for the *ifUserData* interface [implemented in version 4.4.47] in the *roDatagramSender* and *roDatagramReceiver* section.

1.4 Added a description of the `SetBackgroundColor()` method in the *roVideoOutput* entry.

1.5 Added a description and listed the possible parameters for `HasFeature()` in the *roDeviceInfo* entry.

1.6 Added object creation parameters for *roAssetPool* and *roSyncPool* (as well as a more comprehensive introduction for *roAssetPool*)

1.7 Revised description of the `GetResponseCode()` method in the *roUrlEvent* entry.

1.8 Changed the formatting of all example scripts to make them easier to distinguish from definitions and other explanatory language.

March 14, 2013

2.1 Added a description of the `JoinMulticastGroup()`method [implemented in version 4.4.62] to the *roDatagramSocket* entry.

2.2 Added a description of the `EnableScanDebug()`method [implemented in version 4.4.62] to the *roChannelManager* entry.

March 27, 2013

3.1 Added descriptions of `SetAppCacheDir()` and `SetAppCacheSize()` methods [implemented in version 4.4.71] to the *roHtmlWidget* entry.

3.2 Revised *roVideoPlayer* entry to include information about new channel scanning functionality.

April 8, 2013

4.1 Removed uses of “CF:” (compact flash) and “ATA:” directories from example scripts in favor of “SD:”

4.2 Added description of `GetEdidIdentity()` [implemented in 4.4.73] to the *roVideoMode* entry.